数字逻辑与微处理器课程设计报告



任课教师: **王老师、葛老师、冯老师**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | **周磊** | 班级 | **2017211110** |
| 专业 | **通信工程** | 学号 | **2017210494** |
| 组号 | 12 | 学院 | **信息与通信工程学院** |

**实验题目：16位五级流水线cpu设计**

目录

[数字逻辑与微处理器课程设计报告 1](#_Toc27640871)

[一、 设计目的 3](#_Toc27640872)

[二、 设计方案 3](#_Toc27640873)

[三、 指令系统 5](#_Toc27640874)

[四、 主要功能部件设计 6](#_Toc27640875)

[1、PC模块：PC\_reg.v 6](#_Toc27640876)

[2、IF/ID模块：if\_id.v 7](#_Toc27640877)

[3、ID模块：id.v 8](#_Toc27640878)

[4、ID/EX模块：id\_ex.v 10](#_Toc27640879)

[5、Regfile模块：regfile.v 11](#_Toc27640880)

[6、EX模块：ex.v 12](#_Toc27640881)

[7、EX/MEM模块：ex\_mem.v 13](#_Toc27640882)

[8、MEM模块：mem.v 14](#_Toc27640883)

[9、MEM/WB模块：mem\_wb.v 15](#_Toc27640884)

[10、除法（DIV）模块：divide.v 16](#_Toc27640885)

[11、CTRL模块：ctrl.v 17](#_Toc27640886)

[12、状态寄存器模块：flag\_reg.v 17](#_Toc27640887)

[13、HILO模块：hilo\_reg.v 18](#_Toc27640888)

[五、 程序设计 18](#_Toc27640889)

[1、define.v 18](#_Toc27640890)

[2、pc\_reg.v 22](#_Toc27640891)

[3、if\_id.v 23](#_Toc27640892)

[4、id.v 25](#_Toc27640893)

[5、id\_ex.v 39](#_Toc27640894)

[6、ex.v 41](#_Toc27640895)

[7、ex\_mem.v 50](#_Toc27640896)

[8、mem.v 52](#_Toc27640897)

[9、mem\_wb.v 54](#_Toc27640898)

[10、regfile.v 55](#_Toc27640899)

[11、divide.v 57](#_Toc27640900)

[12、hilo\_reg.v 59](#_Toc27640901)

[13、flag\_reg.v 60](#_Toc27640902)

[14、ctrl.v 61](#_Toc27640903)

[15、inst\_rom.v(仿真用) 61](#_Toc27640904)

[16、top\_test.v（仿真） 62](#_Toc27640905)

[17、cpu\_try\_tb.v(testbench) 63](#_Toc27640906)

[18、cpu\_try\_rule.txt 64](#_Toc27640907)

[19、inst.txt 65](#_Toc27640908)

[六、 遇到的问题及解决方法 66](#_Toc27640909)

[1、调试困难的问题 66](#_Toc27640910)

[2、Inout端口没正确处理 70](#_Toc27640911)

[3、EX执行模块输出混乱 70](#_Toc27640912)

[4、对imm的控制是否输出 70](#_Toc27640913)

[5、Load，Store 指令 70](#_Toc27640914)

[6、处理Hazard 72](#_Toc27640915)

[七、课程设计体会及建议 74](#_Toc27640916)

[八、分工 75](#_Toc27640917)

1. 设计目的

1、利用硬件描述语言完成基于FPGA的微处理器的设计与实现，应用数字逻辑和计算机原理所学的理论知识，设计并实现一个具有相对完整指令系统的微处理器，建立起计算机整机系统的概念，掌握计算机设计的基本方法，培养科学的系统分析和解决实际问题的工作能力。

2、通过实验过程，进一步深化学习利用硬件描述语言基于FPGA的实现硬件电路的设计和仿真调试方法，培养对计算机硬件系统设计、调试和运行维护等多方面的技能，同时训练工程设计能力和对新技术新方法的使用与掌握能力。

3、设计一个在硬件端时钟能达到极限时，速度性能较好的五级流水线CPU系统。故本CPU设计以追求速度性能为主要目的。

1. 设计方案

此下为按步骤进行。

1、实现MIPS的五级流水线的逻辑架构。

（1） IF 级：取指令级。从内存中读取指令，并在下一个时钟上升沿到来时把指令送到 ID 级的指令缓冲器中。

（2） ID 级：指令译码器。对 IF 级的指令进行译码，并将控制信号以及运算需要的源操作数送到EX级。

（3） EX 级：执行级。在此过程中，执行运算操作，更新标志位，并将其传到相应的缓冲寄存器；或者产生存储数据的使能信号，同时将要直接储存的数据内容传到MEM级。在下一个时钟上升沿到来前把指令送到 MEM 级的指令缓冲器中。

（4） MEM级：访存级。若送到MEM级的指令为访存指令，则在此阶段进行访存操作。除此之外，还将前级或此级要回写到寄存器的数据在下一次时钟上升沿时送到WB级。

（5） WB级：回写级。对于要写入寄存器的指令，在此阶段将数据写入寄存器。

2、实现流水线结构中的基本指令。

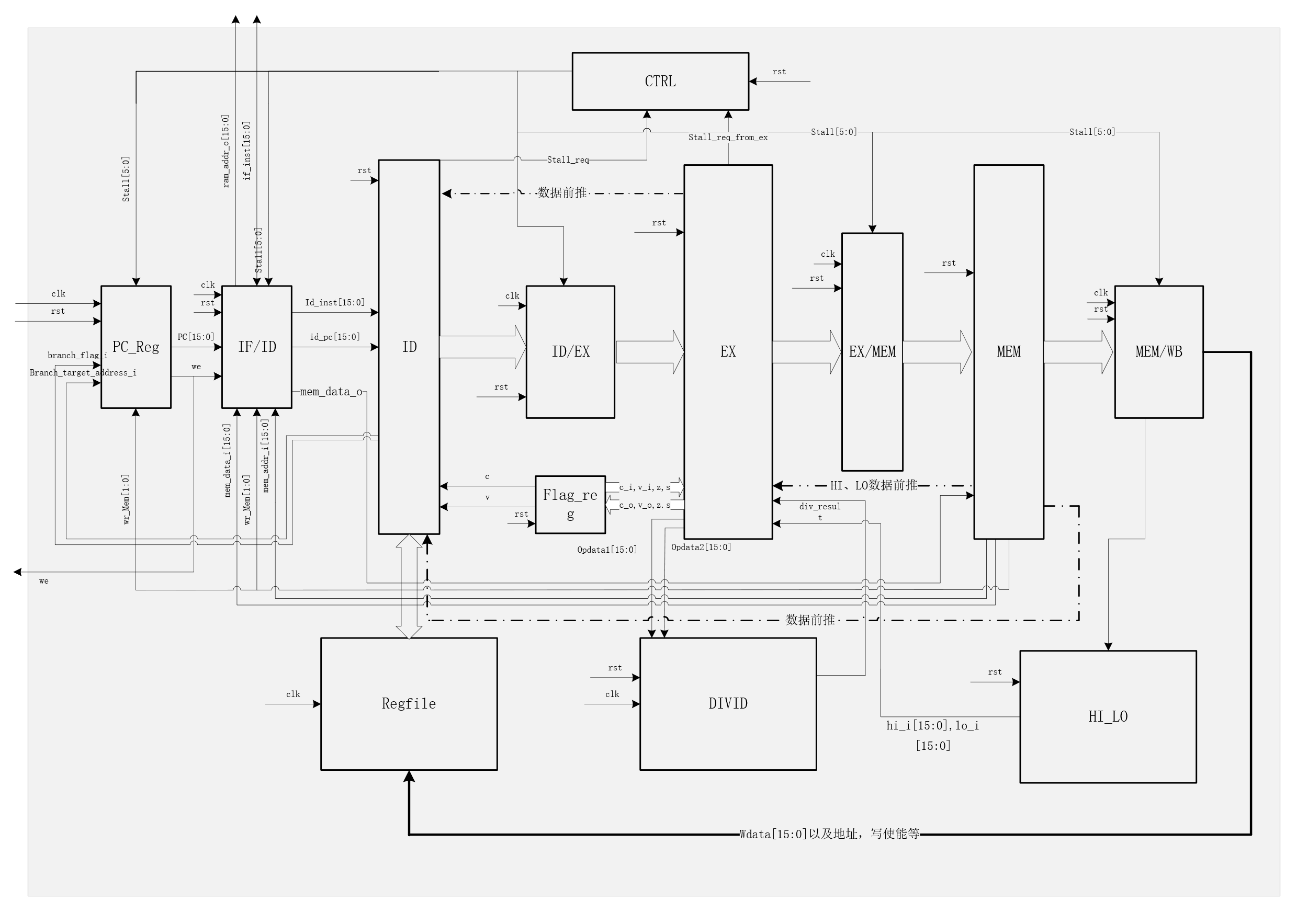
3、添加HI、LO寄存器和与HI、LO寄存器有关的指令，并以此实现乘法指令。

4、添加流水线暂停机制。

5、根据流水线暂停机制添加除法指令。

6、添加跳转指令。

7、添加load、store指令。





1. 指令系统

|  |  |  |  |
| --- | --- | --- | --- |
| INST\_1  (基本逻辑、加减运算) | ADD DR, SR | 0000 0000 | DR+SR->DR，更新标志位 |
| SUB DR, SR | 0000 0001 | DR-SR->DR 更新标志位 |
| DEC DR | 0000 0010 | DR-1→DR 更新标志位 |
| INC DR | 0000 0011 | DR+1→DR 更新标志位 |
| CMP DR, SR | 0000 0110 | DR-SR 比较 |
| AND DR, SR | 0000 0111 | DR and SR→DR 逻辑与 |
| OR DR, SR | 0000 1000 | DR or SR→DR 逻辑或 |
| NOT DR | 0000 1001 | ~DR →DR 逻辑非 |
| XOR DR, SR | 0000 1010 | DR xor SR →DR 异或 |
| TEST DR, SR | 0000 1011 | DR and SR 测试 |
| SHL DR | 0000 1100 | 逻辑左移,最低位补0,最高位移入C |
| SHR DR | 0000 1101 | 逻辑右移,最高位补0,最低位移入C |
| SAR DR | 0000 1110 | 算术右移,最高位右移,同时再用自身的值填入 |
| INST\_2  （R15立即数） | LOADH IMM | 0010 0000 | IMM →R15高8位 |
| LOADL IMM | 0010 0001 | IMM →R15低8位 |
| INST\_3  （LOAD/STORE） | LOAD DR, SR | 0100 0001 | [SR]→DR |
| STORE DR, SR | 0100 0010 | SR →[DR] |
| INST\_4  （NOP） | NOP | 1100 0000 | 空操作 |
| INST\_5  （MOVE、与HI、LO寄存器） | MOVE DR, SR | 0011 0000 | SR →DR |
| MOVZ DR, SR | 0011 0001 | SR →DR,如果DR内为0 |
| MOVN DR, SR | 0011 0010 | SR →DR 如果DR不为0 |
| MFHI DR | 0011 0011 | HI →DR |
| MTHI SR | 0011 0100 | SR →HI |
| MFLO DR | 0011 0101 | LO →DR |
| MTLO SR | 0011 0110 | SR →LO |
| INST\_6  （乘除指令） | MUL DR, SR | 0101 0000 | DR\*SR→{HI,LO} |
| MULU DR, SR | 0101 0001 | DR\*SR→{HI,LO}(无符号运算) |
| DIV DR, SR | 0101 0010 | DR/SR→{HI,LO} |
| DIVU DR, SR | 0101 0011 | DR/SR→{HI,LO} |
| INST\_7  （跳转指令） | J IMM | 0001 0000 | J → PC+IMM |
| JC IMM | 0001 0001 | J → PC+IMM C=1 |
| JNC IMM | 0001 0010 | J → PC+IMM C=0 |
| JZ IMM | 0001 0011 | J → PC+IMM Z=1 |
| JNZ IMM | 0001 0100 | J → PC+IMM Z=0 |

1. 主要功能部件设计

1、PC模块：PC\_reg.v

PC模块的作用的是给出指令地址，以及后面分支跳转时跳转地址的赋值。当流水线不在暂停状态，以及访存阶段没有LOAD、STORE等访存指令时，PC每个时钟加1(实验箱)，个人在仿真时，由于写了一个Ram模块进行仿真，故每回合加2。

1、在流水线暂停时，PC保持不变，直到流水线暂停结束。

2、在load、store指令时，PC根据wr\_Mem信号保持不变一个周期，并根据wr\_Mem[1:0]判断：

00->LOAD指令,we->读

01->STORE指令,we->写

1x->访存阶段无访存指令执行,we->读。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| clk | wire | Input | 时钟 |
| rst | 复位 |
| wr\_Mem[1:0] | 访存信号 |
| stall[5:0] | 流水线暂停信号 |
| branch\_flag\_i | 分支跳转信号 |
| branch\_target\_address\_i[15:0] | 跳转的目的地址 |
| pc[15:0] | Reg | Output | pc寄存器 |
| we | wire | RAM读写控制 |

2、IF/ID模块：if\_id.v

IF/ID模块的作用：

此模块直接与数据总线和地址总线相连。

1、暂时保存取指阶段取得的指令，以及对应的指令地址，并在下一个时钟上升沿传递到译码阶段。

2、由wr\_Mem[1:0]判断是否访存阶段有LOAD、STORE指令进行读写操作。

注：we信号由pc模块产生，并且we信号传输过程始终为wire型。

wr\_Mem[1:0]信号由Mem模块产生，Mem模块为时序模块，但无需clk触发，故在Mem执行过程中， 产生wr\_Mem信号，送到pc模块和if\_id模块，在pc模块，pc在clk上升沿时判断wr\_Mem并进行pc的赋值（如保持不变）。

而we在pc中是wire信号，

而在if\_id模块取指组合逻辑，但在赋予输出寄存器时，则是在时钟上升沿，即当访存阶段发出访存信号时，pc模块，和if\_id模块的pc和取出的指令已经赋给寄存器，而此刻we信号在时钟内进行改变已经对当前周期的执行不会造成影响。而在下个周期则开始进行pc模块保持不变，因为此周期用来进行访存操作，占用了地址和数据总线，导致无取指进行，延后到下一周期。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| clk | wire | Input | 时钟 |
| rst | 复位 |
| if\_pc[15:0] | 取指阶段取得的指令对应的地址 |
| stall[5:0] | 流水线暂停信号 |
| wr\_Mem[1:0] | 访存信号 |
| mem\_addr\_i[15:0] | 要访存的地址 |
| mem\_data\_i[15:0] | 需要store的数据 |
| we | 对Ram的读写控制 |
| if\_inst[15:0] | wire | inout | 取指阶段取得的指令 |
| ram\_addr\_o[15:0] | wire | output | Ram对应的地址 |
| id\_pc[15:0] | reg | 译码阶段的指令对应的地址 |
| id\_inst[15:0] | 译码阶段的指令对 |
| mem\_data\_o[15:0] | load出的数据 |

3、ID模块：id.v

ID模块的作用是对指令进行译码，并得到最终运算的类型、子类型、源操作数1、源操作数2、要写入的目的寄存器等信息，其中运算类型指的是逻辑运算、移位运算、算术运算等，子类型指的是更加详细的运算类型，比如运算类型为逻辑运算时，运算子类型可以为逻辑“或”运算等。

1、ID模块中还包括了imm立即数作为源操作数的问题。模块中具体实现为，添加immenable使能信号。

当reg2\_read\_o = 1’b0 且immenable=1’b1时，imm充当源操作数2参与运算，传递到下一级。

2、ID模块中还有数据相关问题的解决。采用了数据前推的方法，检测执行阶段、访存阶段是否有写入寄存器，并将其写入目的寄存器地址与当前译码阶段要读的源寄存器地址进行比较，如果相同，则采用最新的值，即优先级为EX级MEM级。

3、延迟槽功能。当采用跳转指令时，ID阶段将branch\_address（跳转分支地址）赋给pc阶段，但势必中间会有一条指令（IF/ID模块中）会被浪费。在此系统中，规定此指令为延迟槽指令，不会中断执行。故将问题交给编译层解决，从而保证硬件层不会有资源浪费。主要是解决Hazard控制相关的问题。

4、link\_address\_o-跳转链接地址。此为待拓展功能。目的是实现B指令，即JUMP指令结束后，能够返回原先指令。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| rst | wire | Input | 复位 |
| pc\_i[15:0] | 译码阶段指令对应的地址 |
| inst\_i[15:0] | 译码阶段的指令 |
| reg1\_data\_i[15:0] | 第一个读寄存器端口的输入 |
| reg2\_data\_i[15:0] | 第二个读寄存器端口的输入 |
| ex\_wreg\_i | 处于执行阶段的指令是否有要写入的目的寄存器 |
| ex\_wd\_i[3:0] | 处于执行阶段的指令要写入的目的寄存器地址 |
| ex\_wdata[15:0] | 处于执行阶段的指令要写入目的寄存器的数据 |
| mem\_wreg\_i | 处于访存阶段的指令是否有要写入的目的寄存器 |
| mem\_wd\_i[3:0] | 处于访存阶段的指令要写入的目的寄存器地址 |
| mem\_wdata[15:0] | 处于访存阶段的指令要写入目的寄存器的数据 |
| id\_c | 输入的c标志位 |
| id\_z | 输入的z标志位 |
| reg1\_read\_o | reg | output | 第一个读寄存器端口的读使能信号 |
| reg2\_read\_o | 第二个读寄存器端口的读使能信号 |
| reg1\_addr\_o[3:0] | 第一个读寄存器端口的读地址信号 |
| reg2\_addr\_o[3:0] | 第二个读寄存器端口的读地址信号 |
| aluop\_o[7:0] | 译码阶段指令要进行运算的子类型 |
| alusel\_o[2:0] | 译码阶段指令要进行运算的类型 |
| reg1\_o[15:0] | 译码阶段指令要进行运算的源操作数1 |
| reg2\_o[15:0] | 译码阶段指令要进行运算的源操作数2 |
| wd\_o[3:0] | 译码阶段的指令要写入的目的寄存器地址 |
| wreg\_o | 译码阶段的指令是否有要写入的目的寄存器 |
| flag\_upd\_o | 译码阶段的指令是否要更新标志位 |
| next\_inst\_in\_delayslot\_o | 下一条进入译码阶段的指令是否位于延迟槽 |
| branch\_flag\_o | 是否发生转移 |
| branch\_target\_address\_o[15:0] | 转移到的目标地址 |
| link\_addr\_o[15:0] | 转移指令要保存的返回地址 |
| is\_in\_delayslot\_o | 当前处于译码阶段的指令是否位于延迟槽 |
| is\_in\_delayslot\_i | wire | input | 当前处于译码阶段的指令是否位于延迟槽 |

4、ID/EX模块：id\_ex.v

ID/EX模块的作用是将译码阶段取得的运算类型、源操作数、要写的目的寄存器地址等结果，在下一个时钟传递到流水线执行阶段。

此阶段还进行延迟槽指令状态的交替更新功能。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| clk | wire | Input | 时钟 |
| rst | 复位 |
| stall[5:0] | 流水线暂停信号 |
| id\_aluop[7:0] | 译码阶段指令要进行运算的子类型 |
| id\_alusel[2:0] | 译码阶段指令要进行运算的类型 |
| id\_reg1[15:0] | 译码阶段的指令运算的源操作数1 |
| id\_reg2[15:0] | 译码阶段的指令运算的源操作数2 |
| id\_wd[3:0] | 译码阶段的指令要写入的目的寄存器地址 |
| id\_wreg | 译码阶段的指令是否有要写入的目的寄存器 |
| id\_flag\_upd | 译码阶段的指令是否要更新标志位 |
| id\_is\_in\_delayslot | 当前处于译码阶段的指令是否位于延迟槽 |
| id\_link\_address[15:0] | 处于译码阶段的指令要保存的返回地址 |
| next\_inst\_in\_delayslot\_i | 下一条进入译码阶段的指令是否位于延迟槽 |
| ex\_is\_in\_delayslot | reg | output | 当前处于执行阶段的指令是否位于延迟槽 |
| ex\_link\_address[15:0] | 处于执行阶段的指令要保存的返回地址 |
| is\_in\_delayslot\_o | 当前处于译码阶段的指令是否位于延迟槽 |
| ex\_aluop[7:0] | 执行阶段指令的运算子类型 |
| ex\_alusel[2:0] | 执行阶段指令的运算类型 |
| ex\_reg1[15:0] | 执行阶段指令运算的源操作数1 |
| ex\_reg2[15:0] | 执行阶段指令运算的源操作数2 |
| ex\_wd[3:0] | 执行阶段指令要写入的目的寄存器地址 |
| ex\_wreg | 执行阶段指令是否要写入目的寄存器 |
| ex\_flag\_upd | 执行阶段指令是否要更新标志位 |

5、Regfile模块：regfile.v

Regfile模块实现了16个16位通用寄存器，可以同时进行两个寄存器的读操作和一个寄存器的写操作。

注：写入数据需要clk上升沿触发，而读数据无需时钟触发。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| clk | wire | Input | 时钟 |
| rst | 复位 |
| sel[1:0] | 选择信号（实验箱） |
| reg\_sel[3:0] | 信号灯选择寄存器信号 |
| we | 写使能信号 |
| waddr[3:0] | 要写入的寄存器地址 |
| wdata[15:0] | 要写入的数据 |
| re1 | 第一个读寄存器端口的读使能信号 |
| raddr1[3:0] | 第一个读寄存器端口要读取的寄存器地址 |
| rdata1[15:0] | reg | Output | 第一个读寄存器端口输出的寄存器值 |
| re2 | wire | Input | 第二个读寄存器端口的读使能信号 |
| raddr2[3:0] | 第二个读寄存器端口尧都区的寄存器地址 |
| rdata2[15:0] | reg | Output | 第二个读寄存器端口输出的寄存器值 |
| reg\_data[15:0] | wire | 向实验箱输出的寄存器值 |

6、EX模块：ex.v

EX模块根据输入的运算信息进行数据运算，并给出最终的运算结果及是否写入通用寄存器或HI、LO寄存器等信息。

EX模块中对应不同的运算类型有不同的缓存寄存器来记录结果，如result\_sum保存加减法结果，logicout保存逻辑运算结果，而mulres保存乘法运算结果，moveres保存移动运算结果。

最终根据指令运算类型，将其中的一个赋给wdata\_o,即最终要写入目的寄存器的值。

而在EX模块中，也进行标志位的更新，如在ADD，SUB指令中，存在C，V，Z的更新。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| rst | wire | Input | 复位 |
| aluop\_i[7:0] | 执行阶段要进行运算的子类型 |
| alusel\_i[2:0] | 执行阶段要进行运算的类型 |
| reg1\_i[15:0] | 参与运算的源操作数1 |
| reg2\_i[15:0] | 参与运算的源操作数2 |
| wd\_i[3:0] | 指令执行要写入的目的寄存器地址 |
| wreg\_i | 是否有要写入的目的寄存器 |
| c\_i | 输入的c |
| v\_i | 输入的v |
| z\_i | 输入的z |
| s\_i | 输入的s |
| ex\_flag\_upd\_i | 执行阶段的指令是否需要更新标志位 |
| c\_o | reg | Output | 输出的c |
| v\_o | 输出的v |
| z\_o | 输出的z |
| s\_o | 输出的s |
| wd\_o[3:0] | 执行阶段指令最终要写入的目的寄存器地址 |
| wreg\_o | 执行阶段的指令最总是否有要写入的目的寄存器 |
| wdata\_o[15:0] | 执行阶段的指令最终要写入目的寄存器的值 |
| hi\_i[15:0] | wire | Input | HILO模块给出的HI寄存器值 |
| lo\_i[15:0] | HILO模块给出的LO寄存器值 |
| wb\_hi\_i[15:0] | 处于回写阶段的指令要写入HI的值 |
| wb\_lo\_i[15:0] | 处于回写阶段的指令要写入LO的值 |
| wb\_whilo\_i | 处于回写阶段的指令是否要写HI、LO寄存器 |
| mem\_hi\_i[15:0] | 处于访存阶段的指令要写入HI的值 |
| mem\_lo\_i[15:0] | 处于访存阶段的指令要写入LO的值 |
| mem\_whilo\_i | 处于访存阶段的指令是否要写HI、LO寄存器 |
| whilo\_o | reg | Output | 执行阶段的指令是否要写HI、LO寄存器 |
| hi\_o[15:0] | 执行阶段的指令要写入HI的值 |
| lo\_o[15:0] | 执行阶段的指令要写入LO的值 |
| div\_result\_i[31:0] | wire | Input | 除法运算结果 |
| div\_ready\_i | 除法运算是否结束 |
| div\_opdata1\_o[15:0] | reg | Output | 被除数 |
| div\_opdata2\_o[15:0] | 除数 |
| div\_start\_o | 是否开始除法运算 |
| signed\_div\_o | 是否是有符号除法 |
| stallreq | 执行阶段的流水线暂停请求信号 |
| link\_address\_i[15:0] | wire | Input | 处于执行阶段的指令要保存的返回地址 |
| is\_in\_delayslot\_i | 当前处于执行阶段的指令是否位于延迟槽 |
| aluop\_o[7:0] | reg | Output | 执行阶段的运算子类型 |
| mem\_addr\_o[15:0] | 执行阶段指令最终要进行访存的地址 |
| data\_store[15:0] | 执行阶段指令最终要store的数据 |

7、EX/MEM模块：ex\_mem.v

EX/MEM模块将执行阶段取得的运算结果在下一个时钟上升沿传递到流水线访存阶段。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| clk | wire | Input | 时钟 |
| rst | 复位 |
| stall[5:0] | 流水线暂停信号 |
| ex\_wd[3:0] | 执行阶段指令执行后要写入的目的寄存器地址 |
| ex\_wreg | 执行阶段的指令执行后是否有要写入的目的寄存器 |
| ex\_wdata[15:0] | 执行阶段的指令执行后要写入目的寄存器的值 |
| ex\_hi[15:0] | 执行阶段的指令要写入HI的值 |
| ex\_lo[15:0] | 执行阶段的指令要写入LO的值 |
| ex\_whilo | 执行阶段的指令是否要写HI、LO寄存器 |
| ex\_aluop[7:0] | 执行阶段指令的子类型 |
| ex\_mem\_addr[15:0] | 执行阶段指令最终要访存的地址 |
| ex\_data\_store[15:0] | 执行阶段指令最终要store的数据 |
| mem\_wd[3:0] | reg | Output | 访存阶段指令要写入的目的寄存器地址 |
| mem\_wreg | 访存阶段的指令是否有要写入的目的寄存器 |
| mem\_wdata[15:0] | 访存阶段的指令要写入目的寄存器的值 |
| mem\_hi[15:0] | 访存阶段的指令要写入HI的值 |
| mem\_lo[15:0] | 访存阶段的指令要写入LO的值 |
| mem\_whilo | 访存阶段的指令是否要写HI、LO寄存器 |
| mem\_aluop[7:0] | 访存阶段指令的子类型 |
| mem\_mem\_addr[15:0] | 访存阶段指令要进行访存的地址 |
| mem\_data\_store[15:0] | 访存阶段指令要store的数据 |

8、MEM模块：mem.v

MEM模块在访存指令时产生访存控制信号wr\_Mem，将访存地址和store的数据送到IF/ID模块，或者从IF/ID模块读取LOAD出的数据并送到下一级回写阶段。而在其他指令时则只是将信息传递到下一级。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| rst | wire | Input | 复位 |
| wd\_i[3:0] | 访存阶段指令要写入的目的寄存器地址 |
| wreg\_i | 访存阶段指令是否有要写入的目的寄存器 |
| wdata\_i[15:0] | 访存阶段指令要写入目的寄存器的值 |
| hi\_i[15:0] | 访存阶段的指令要写入HI的值 |
| lo\_i[15:0] | 访存阶段的指令要写入LO的值 |
| whilo\_i | 访存阶段的指令是否要写HI、LO寄存器 |
| aluop\_i[7:0] | 访存阶段指令的子类型 |
| mem\_addr\_i[15:0] | 访存阶段要访存的地址 |
| mem\_data\_store[15:0] | 访存阶段要store的数据 |
| mem\_data\_i[15:0] | 访存阶段load的数据 |
| wd\_o[3:0] | reg | Output | 访存阶段指令最终要写入的目的寄存器 |
| wreg\_o | 访存阶段指令最终是否有要写入的目的寄存器 |
| wdata\_o[15:0] | 访存阶段指令最终要写入目的寄存器的值 |
| hi\_o[15:0] | 访存阶段的指令最终要写入HI的值 |
| lo\_o[15:0] | 访存阶段的指令最终要写入LO的值 |
| whilo\_o | 访存阶段的指令最终是否要写HI、LO寄存器 |
| mem\_addr\_o[15:0] | 访存阶段要访存的地址 |
| mem\_data\_o[15:0] | 访存阶段写入内存（store）的数据 |
| wr\_Mem[1:0] | 访存的信号(00 load 01 store) |

9、MEM/WB模块：mem\_wb.v

MEM/WB模块的作用是将访存阶段的运算结果，在下一个时钟传递到回写阶段。

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名称 | 变量类型 | 方向 | 说明 |
| clk | wire | Input | 时钟 |
| rst | 复位 |
| stall[5:0] | 流水线暂停信号 |
| mem\_wd[3:0] | 访存阶段的指令最终要写入的目的寄存器地址 |
| mem\_wreg | 访存阶段的指令最终是否有要写入的目的寄存器 |
| mem\_wdata[15:0] | 访存阶段的指令最终要写入目的寄存器的值 |
| mem\_hi[15:0] | 访存阶段的指令要写入HI的值 |
| mem\_lo[15:0] | 访存阶段的指令要写入LO的值 |
| mem\_whilo | 访存阶段的指令是否要写HI、LO寄存器 |
| wb\_wd[3:0] | reg | Output | 回写阶段的指令要写入的目的寄存器地址 |
| wb\_wreg | 回写阶段指令是否有要写入的目的寄存器 |
| wb\_wdata[15:0] | 回写阶段指令要写入目的寄存器的值 |
| wb\_hi[15:0] | 回写阶段的指令要写入HI的值 |
| wb\_lo[15:0] | 回写阶段的指令要写入LO的值 |
| wb\_whilo | 回写阶段的指令是否要写HI、LO寄存器 |

10、除法（DIV）模块：divide.v

采用试商法实现除法。

输入被除数opdata1\_i[15:0],输入除数opdata2\_i[15:0],而用32位的dividend[31:0]作为迭代缓冲寄存器，并在最后结束时，高16位输出余数，低16位输出商。

1、将被除数赋给dividend低16位，并将dividend高16位置0。

2、判断的最高位是否为1，为1则代表出现借位，则将dividend左移一位，并将0赋值给dividend最低位。若为0，则代表此次除法试商结果为1，则将减法结果的低16位赋给dividend高16位，并将低15位赋值给dividend[15:1]，将最后一位赋值为1。

3、判断cnt及annul\_i等条件并执行相应的迭代或跳转步骤。

除法中的状态转移图：



|  |  |  |  |
| --- | --- | --- | --- |
| 变量名称 | 变量类型 | 方向 | 说明 |
| clk | wire | Input | 时钟 |
| rst | 复位 |
| signed\_div\_i | 是否为有符号除法 |
| opdata1\_i[15:0] | 被除数 |
| opdata2\_i[15:0] | 除数 |
| start\_i | 是否开始除法运算 |
| annul\_i | 是否取消除法运算 |
| result\_o[31:0] | reg | Output | 除法运算结果 |
| ready\_o | 除法运算是否结束 |

11、CTRL模块：ctrl.v

实现流水线暂停机制。输出信号stall是一个宽度为6的信号，其含义如下：

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[2]表示流水线译码阶段是否暂停，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。

当处于流水线执行阶段的指令请求暂停时，则其后的阶段继续，而其前的阶段暂停，stall为6’b001111。

当处于流水线译码阶段的指令请求暂停时，stall为6’b000111。其余情况stall设置为6’b000000，表示不暂停流水线。

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名称 | 接口类型 | 方向 | 说明 |
| rst | wire | input | 复位 |
| stallreq\_from\_id | 来自id级的暂停请求 |
| stallreq\_from\_ex | 来自ex级的暂停请求（除法指令） |
| stall[5:0] | reg | output | 流水线暂停信号 |

12、状态寄存器模块：flag\_reg.v

在每一个时钟上升沿更新标志位寄存器。

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名称 | 变量类型 | 方向 | 说明 |
| clk | wire | input | 时钟 |
| rst | 复位 |
| c\_i | 写入的c |
| z\_i | 写入的z |
| v\_i | 写入的v |
| s\_i | 写入的s |
| c\_o | reg | output | 输出的c |
| z\_o | 输出的z |
| v\_o | 输出的v |
| s\_o | 输出的s |

13、HILO模块：hilo\_reg.v

寄存器模块，主要储存乘除法结果，并配以MFHI、MTHI、MFLO、MTLO等指令实现结果的存取。

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名称 | 变量类型 | 方向 | 说明 |
| clk | wire | input | 时钟 |
| rst | 复位 |
| we | 写使能信号 |
| hi\_i[15:0] | 要写入的hi |
| lo\_i[15:0] | 要写入的lo |
| hi\_o[15:0] | reg | output | 输出hi |
| lo\_o[15:0] | 输出lo |

1. 程序设计

1、define.v

进行程序设计中宏的定义。

//全局宏定义

`define RstEnable 1'b0 //复位信号有效

`define RstDisable 1'b1 //复位信号无效

`define ZeroHalfWord 16'h0000

`define WriteEnable 1'b1

`define WriteDisable 1'b0

`define ReadEnable 1'b1

`define ReadDisable 1'b0

`define AluOpBus 7**:**0 //译码阶段的输出aluop\_o的宽度

`define AluSelBus 2**:**0 //译码阶段的输出alusel\_o的宽度

`define InstValid 1'b1

`define InstInvalid 1'b0

`define True\_v 1'b1

`define False\_v 1'b0

`define ChipWrite 1'b0

`define ChipRead 1'b1

`define Stop 1'b1

`define NoStop 1'b0

/\*PC指令寄存器有关的宏定义\*/

`define InstAddrBus 15**:**0

`define InstBus 15**:**0

`define NoFlagUpd 1'b0

//与跳转有关

`define Branch 1'b1

`define NotBranch 1'b0 //不跳转

`define InDelaySlot 1'b1

`define NotInDelaySlot 1'b0

/\*通用寄存器组有关的宏定义\*/

`define RegAddrBus 3**:**0 //通用寄存器组的地址线宽度，因为定义16个寄存器

`define RegBus 15**:**0 //通用寄存器组的数据线宽度

`define RegWidth 16 //通用寄存器宽度

`define DoubleRegWidth 32 //两倍的通用寄存器的宽度

`define DoubleRegBus 31**:**0 //两倍的通用寄存器的数据线宽度

`define RegNum 16 //通用寄存器的数量

`define RegNumLog2 4 //通用寄存器寻址所需要的地址位数

`define NOPRegAddr 4'b0000

`define DataAddrBus 15**:**0

`define DataBus 15**:**0

/\*指令执行的宏定义\*/

`define INST\_1 4'b0000 //基本加减和运算

`define INST\_2 4'b0010 //立即数

`define INST\_3 4'b0100 //堆栈指针 // movei load /store

`define INST\_4 4'b1100 //NOP

`define INST\_5 4'b0011 //HILO

`define INST\_6 4'b0101 //乘除 乘+

`define INST\_7 4'b0001 //跳转

//0000 基本的逻辑指令和运算指令

`define EXE\_ADD 4'b0000

`define EXE\_SUB 4'b0001 //0000

`define EXE\_DEC 4'b0010

`define EXE\_INC 4'b0011

`define EXE\_CMP 4'b0110

`define EXE\_AND 4'b0111

`define EXE\_OR 4'b1000

`define EXE\_NOT 4'b1001

`define EXE\_XOR 4'b1010

`define EXE\_TEST 4'b1011

`define EXE\_SHL 4'b1100

`define EXE\_SHR 4'b1101

`define EXE\_SAR 4'b1110

//0001跳转指令

`define EXE\_J 4'b0000

`define EXE\_JC 4'b0001

`define EXE\_JNC 4'b0010

`define EXE\_JZ 4'b0011

`define EXE\_JNZ 4'b0100

//0100堆栈指令

//0011 hilo指令

`define EXE\_MOVE 4'b0000

`define EXE\_MOVZ 4'b0001

`define EXE\_MOVN 4'b0010

`define EXE\_MFHI 4'b0011

`define EXE\_MTHI 4'b0100

`define EXE\_MFLO 4'b0101

`define EXE\_MTLO 4'b0110

//0010

`define EXE\_LOADH 4'b0000

`define EXE\_LOADL 4'h0001 //0010

//0100

`define EXE\_LOAD 4'b0001

`define EXE\_STORE 4'b0010

//1100

`define EXE\_NOP 4'b0000

//0101 乘除

`define EXE\_MUL 4'b0000 //有符号乘法

`define EXE\_MULU 4'b0001 //无符号乘法

`define EXE\_DIV 4'b0010

`define EXE\_DIVU 4'b0011 //结果 LO 余数HI

//ALUOP

`define EXE\_ADD\_OP 8'b00000000

`define EXE\_SUB\_OP 8'b00000010

`define EXE\_CMP\_OP 8'b00000110

`define EXE\_AND\_OP 8'b00000111

`define EXE\_OR\_OP 8'b00001000

`define EXE\_NOT\_OP 8'b00001001

`define EXE\_XOR\_OP 8'b00001010

`define EXE\_TEST\_OP 8'b00001011

`define EXE\_SHL\_OP 8'b00001100

`define EXE\_SHR\_OP 8'b00001101

`define EXE\_SAR\_OP 8'b00001110

`define EXE\_NOP\_OP 8'b11000000

//跳转

`define EXE\_J\_OP 8'b00010000

`define EXE\_JC\_OP 8'b00010001

`define EXE\_JRNC 8'b00010010

`define EXE\_JRZ 8'b00010011

`define EXE\_JRNZ 8'b00010100

//0100 堆栈 /load /store /movei

`define EXE\_LOAD\_OP 8'b01000001

`define EXE\_STORE\_OP 8'b01000010

`define EXE\_MOVEI 8'b01000011

`define EXE\_PUSH\_OP 8'b01000110

`define EXE\_POP\_OP 8'b01000111

//

`define EXE\_MOVE\_OP 8'b00110000

`define EXE\_MOVZ\_OP 8'b00110001

`define EXE\_MOVN\_OP 8'b00110010

`define EXE\_MFHI\_OP 8'b00110011

`define EXE\_MTHI\_OP 8'b00110100

`define EXE\_MFLO\_OP 8'b00110101

`define EXE\_MTLO\_OP 8'b00110110

//

`define EXE\_MUL\_OP 8'b01010000

`define EXE\_MULU\_OP 8'b01010001

`define EXE\_DIV\_OP 8'b01010010

`define EXE\_DIVU\_OP 8'b01010011

`define EXE\_RES\_LOGIC 3'b001

`define EXE\_RES\_SHIFT 3'b010

`define EXE\_RES\_MOVE 3'b011

`define EXE\_RES\_ARITHMETIC 3'b100

`define EXE\_RES\_MUL 3'b101

`define EXE\_RES\_NOP 3'b000

`define EXE\_RES\_JUMP\_BRANCH 3'b110

`define EXE\_RES\_LOAD\_STORE 3'b111

//与除法器有关的宏

`define DivFree 2'b00

`define DivByZero 2'b01

`define DivOn 2'b10

`define DivEnd 2'b11

`define DivResultReady 1'b1

`define DivResultNotReady 1'b0

`define DivStart 1'b1

`define DivStop 1'b0

// ROM

`define InstNum 1024 //rom实际大小1kb

`define InstMemNumLog2 10//ROM实际使用的地址总线宽度

2、pc\_reg.v

`include"define.v"

**module** pc\_reg**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire[**5**:**0**]** stall**,** //流水线暂停

//应添加 控制we的信号 chip //根据指令的类型

**input** **wire** branch\_flag\_i**,**

**input** **wire** **[**`RegBus**]** branch\_target\_address\_i**,**

//增加 要写内存的数据的地址信号

**output** **reg[**`InstAddrBus**]** pc**,**

//

//input wire[`RegBus] mem\_addr\_i,

**input** **wire[**1**:**0**]** wr\_Mem**,**//信号

//

**output** **wire** we

**);**

**assign** we **=** **(**wr\_Mem**==**2'b01**)?** `ChipWrite**:** `ChipRead**;**

**always@(posedge** clk**)**

**begin**

**if(** rst **==** `RstEnable**)**

**begin**

// we<=`ChipRead; //复位的时候指令存储器禁用

pc**<=**16'h0000**;**

**end**

**else** **begin**

// we<=`ChipRead;

**if(**wr\_Mem**==**2'b00**)** **begin**

// we<=`ChipRead;

pc**<=**pc**;**

**end**

**else** **if(**wr\_Mem**==**2'b01**)** **begin**

// we<=`ChipWrite;

pc**<=**pc**;**

**end**

**else** **if(**stall**[**0**]==**`NoStop**)**

**begin**

**if(**we**==**`ChipRead**)** **begin**

**if(**branch\_flag\_i**==**`Branch**)** **begin**

pc**<=**branch\_target\_address\_i**;end**

**else** **begin**

pc**<=**pc**+**4'h2**;**

**end**

**end**

**else** **if(**we**==**`ChipWrite**)** **begin**

pc**<=**16'h0000**;** //无需修改 ----alu\_out 写地址

**end**

**end**

**end** // 待修改

**end**

**endmodule**

3、if\_id.v

/\*if\_id暂时保存取指阶段获得的指令，以及对应的指令地址，并在下一个时钟传递到译码阶段

\*/

`include"define.v"

**module** if\_id**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire[**5**:**0**]** stall**,**

**input** **wire** **[**`InstAddrBus**]** if\_pc**,** //InstAddrBus 为地址总线宽度

**inout** **wire** **[**`InstBus**]** if\_inst**,** //InstBus为指令总线的宽度 此为输入的数据

**output** **wire** **[**`InstAddrBus**]** ram\_addr\_o**,**

**output** **reg** **[**`InstAddrBus**]** id\_pc**,**

**output** **reg** **[**`InstBus**]** id\_inst**,**

/\*

待增加信号 we 写信号

增加要写的数据 alu\_out

//根据指令类型 更改we

\*/

**input** **wire[**1**:**0**]** wr\_Mem**,**

**input** **wire[**`RegBus**]** mem\_addr\_i**,**

**input** **wire[**`RegBus**]** mem\_data\_i**,** //store

**output** **wire[**`RegBus**]** mem\_data\_o**,** //load

**input** **wire** we

**);**

**reg[**`InstBus**]** mem\_data\_in**;**

**wire[**`InstBus**]** mem\_data\_out**;**

**assign** mem\_data\_out **=** mem\_data\_i**;**

**assign** mem\_data\_o **=** **(**wr\_Mem**==**2'b00**)?** if\_inst**:**16'h0000**;**

**assign** if\_inst **=(**we**==**`ChipRead**)?** 16'hzzzz**:** mem\_data\_i**;**

//always@(posedge clk) begin

// if(wr\_Mem[1]==1'b1) begin

// ram\_addr\_o <= if\_pc;

// end

// else begin

// ram\_addr\_o <=mem\_addr\_i;

// end

//end

**assign** ram\_addr\_o **=(**wr\_Mem**[**1**]==**1'b1**)** **?** if\_pc**:** mem\_addr\_i**;**

**always@(\*)** **begin**

**if(**wr\_Mem**==**2'b00**)begin** //load

// mem\_data\_o<=if\_inst;

mem\_data\_in**<=** `EXE\_NOP\_OP**;**

**end**

**else** **if(**wr\_Mem**==**2'b01**)begin**

// mem\_data\_out<=mem\_data\_i;

mem\_data\_in**<=**`EXE\_NOP\_OP**;**

**end**

**else** **begin**

mem\_data\_in **<=**if\_inst**;**

**end**

**end**

//always@(\*) begin

// if(we==`ChipRead) begin

//

// end

// else if(we==`ChipWrite) begin

// // wr\_mem\_data 用alu运算结果更新mem\_data\_out;

// end

//end

// stall[1]为stop ，stall[2]nostop 则为取指阶段暂停

// stall[1] nostop 取指阶段继续，取得指令进入译码阶段

// 其余情况保持译码阶段的 id\_pc,id\_inst不变。

**always@(posedge** clk**)**

**begin**

**if(**rst**==**`RstEnable**)**

**begin**

id\_pc**<=**`ZeroHalfWord**;** //复位时pc为0

// id\_inst<={`INST\_4,`EXE\_NOP,8'h00}; //复位时候指令为0 空指令

**end**

**else** **if((**stall**[**1**]==**`Stop**&&**stall**[**2**]==**`NoStop**)||(**wr\_Mem**[**1**]==**1'b0**))** **begin**

id\_pc**<=**`ZeroHalfWord**;**

id\_inst**<=**`EXE\_NOP\_OP**;** //取指阶段暂停，向译码阶段传递空指令

**end**

**else** **if(**stall**[**1**]==**`NoStop**)** **begin**

// if(wr\_Mem[1]==1'b0) begin

**if(**we**==**`ChipRead**)**

**begin**

id\_pc**<=**if\_pc**;** //向id传递if阶段的值

id\_inst**<=**mem\_data\_in**;**

**end**

**else** **if(**we**==**`ChipWrite**)** //无影响

**begin**

**end**

**end**

**end**

**endmodule**

4、id.v

`include"define.v"

**module** id**(**

**input** **wire** rst**,**

**input** **wire[**`InstAddrBus**]** pc\_i**,** //译码阶段的指令对应的地址

**input** **wire[**`InstBus**]** inst\_i**,** //译码阶段的指令

//从通用寄存器读取的值

**input** **wire[**`RegBus**]** reg1\_data\_i**,** //寄存器组 读端口1 的输入数据 dr

**input** **wire[**`RegBus**]** reg2\_data\_i**,** //寄存器 2 sr

//写入通用寄存器的值

**output** **reg** reg1\_read\_o**,** //寄存器组 读端口1 的使能信号

**output** **reg** reg2\_read\_o**,** // 2

**output** **reg[**`RegAddrBus**]** reg1\_addr\_o**,** //寄存器组 读端口1的地址选择信号

**output** **reg[**`RegAddrBus**]** reg2\_addr\_o**,**

//送到执行阶段的信息

**output** **reg[**`AluOpBus**]** aluop\_o**,** //译码阶段的指令要进行运算的子类型

**output** **reg[**`AluSelBus**]** alusel\_o**,** //译码阶段的指令要进行运算的类型

**output** **reg[**`RegBus**]** reg1\_o**,** //译码阶段的指令要进行运算的源操作数1

**output** **reg[**`RegBus**]** reg2\_o**,** // 2

**output** **reg[**`RegAddrBus**]** wd\_o**,** //译码阶段的指令要写入的目的寄存器地址

**output** **reg** wreg\_o**,** //译码阶段的指令是否有要写入的目的寄存器

**output** **reg** flag\_upd\_o**,** //是否更新标志位

//解决数据相关问题

**input** **wire** ex\_wreg\_i**,**

**input** **wire[**`RegAddrBus**]** ex\_wd\_i**,**

**input** **wire[**`RegBus**]** ex\_wdata**,**

//

**input** **wire** mem\_wreg\_i**,**

**input** **wire[**`RegAddrBus**]** mem\_wd\_i**,**

**input** **wire[**`RegBus**]** mem\_wdata**,**

**output** **wire** stallreq**,** //load store

//跳转指令

**input** **wire** id\_c**,**

**input** **wire** id\_z**,**

**input** **wire** is\_in\_delayslot\_i**,**

**output** **reg** next\_inst\_in\_delayslot\_o**,**

**output** **reg** branch\_flag\_o**,**

**output** **reg[**`RegBus**]** branch\_target\_address\_o**,**

**output** **reg[**`RegBus**]** link\_addr\_o**,**

**output** **reg** is\_in\_delayslot\_o //如果为1，表示此阶段为延迟槽指令

**);**

**wire[**`RegBus**]** pc\_plus\_4**;** // 硬件 2 仿真 4

**wire[**`RegBus**]** pc\_plus\_2**;** //修改 1 2

**wire[**`RegBus**]** imm\_sll2\_signedext**;**

**assign** pc\_plus\_4 **=** pc\_i **+**4**;** //保存当前译码阶段指令后第2条指令的地址

**assign** pc\_plus\_2 **=** pc\_i **+**2**;** //保存当前译码阶段指令后第一条指令的地址

**assign** stallreq **=** `NoStop**;**

//获得指令的指令码

**wire[**3**:**0**]** op\_type **=** inst\_i**[**15**:**12**];** //操作码前4位

**wire[**3**:**0**]** op **=** inst\_i**[**11**:**8**];**

**wire[**3**:**0**]** op1 **=** inst\_i**[**7**:**4**];**

**wire[**3**:**0**]** op2 **=** inst\_i**[**3**:**0**];**

//保存指令中的立即数

**reg[**`RegBus**]** imm**;**

//指令是否有效

**reg** instvalid**;**

**reg** immenable**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*——1——指令译码\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**always** **@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

aluop\_o **<=** `EXE\_NOP\_OP**;**

alusel\_o **<=** `EXE\_RES\_NOP**;**

wd\_o **<=** `NOPRegAddr**;**

wreg\_o **<=** `WriteDisable**;**

instvalid **<=** `InstValid**;**

reg1\_read\_o **<=** 1'b0**;**

reg2\_read\_o **<=** 1'b0**;**

reg1\_addr\_o **<=** `NOPRegAddr**;**

reg2\_addr\_o **<=** `NOPRegAddr**;**

imm **<=** 16'h0**;**

flag\_upd\_o **<=** `NoFlagUpd**;**

link\_addr\_o **<=** `ZeroHalfWord**;**

branch\_target\_address\_o**<=**`ZeroHalfWord**;**

branch\_flag\_o **<=** `NotBranch**;**

next\_inst\_in\_delayslot\_o **<=**`NotInDelaySlot**;**

**end**

**else** **begin**

aluop\_o **<=** `EXE\_NOP\_OP**;**

alusel\_o **<=** `EXE\_RES\_NOP**;**

wd\_o **<=** inst\_i**[**7**:**4**];** //写入寄存器的序号

wreg\_o **<=** `InstInvalid**;**

flag\_upd\_o **<=** `NoFlagUpd**;**

reg1\_read\_o **<=** 1'b0**;**

reg2\_read\_o **<=** 1'b0**;**

reg1\_addr\_o **<=** inst\_i**[**7**:**4**];** //DR寄存器

reg2\_addr\_o **<=** inst\_i**[**3**:**0**];** //SR寄存器

imm **<=** `ZeroHalfWord**;**

flag\_upd\_o **<=** `NoFlagUpd**;**

link\_addr\_o **<=** `ZeroHalfWord**;**

branch\_target\_address\_o**<=**`ZeroHalfWord**;**

branch\_flag\_o **<=** `NotBranch**;**

next\_inst\_in\_delayslot\_o **<=**`NotInDelaySlot**;**

**case(**op\_type**)**

`INST\_1**:begin** //基本的加减 + 逻辑运算

**case(**op**)**

`EXE\_ADD**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_ADD\_OP**;**

alusel\_o **<=**`EXE\_RES\_ARITHMETIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg2\_read\_o **<=** 1'b1**;**

wd\_o **<=** op1**;**

flag\_upd\_o **<=** 1'b1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_SUB**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_SUB\_OP**;**

alusel\_o **<=**`EXE\_RES\_ARITHMETIC**;**

reg1\_read\_o**<=** 1'b1**;**

reg2\_read\_o**<=** 1'b1**;**

wd\_o **<=** op1**;**

flag\_upd\_o **<=**1'b1**;**

instvalid**<=** `InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_DEC**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_SUB\_OP**;**

alusel\_o **<=**`EXE\_RES\_ARITHMETIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg2\_read\_o **<=** 1'b0**;**

imm **<=** 16'h0001**;**

wd\_o **<=** op1**;**

flag\_upd\_o **<=**1'b1**;**

instvalid **<=** `InstValid**;**

immenable **<=**1'b1**;**

**end**

`EXE\_INC**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_ADD\_OP**;**

alusel\_o **<=**`EXE\_RES\_ARITHMETIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg2\_read\_o **<=** 1'b0**;**

imm **<=** 16'h0001**;**

wd\_o **<=** op1**;**

instvalid **<=**`InstValid**;**

flag\_upd\_o **<=**1'b1**;**

immenable **<=**1'b1**;**

**end**

`EXE\_CMP**:** **begin**

wreg\_o **<=**`WriteDisable**;**

aluop\_o **<=**`EXE\_CMP\_OP**;**

alusel\_o **<=**`EXE\_RES\_ARITHMETIC**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o **<=**1'b1**;**

instvalid**<=**`InstValid**;**

flag\_upd\_o**<=**1'b1**;**

immenable **<=**1'b0**;**

**end**

`EXE\_AND**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_AND\_OP**;**

alusel\_o **<=**`EXE\_RES\_LOGIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg2\_read\_o **<=** 1'b1**;**

wd\_o **<=** op1**;**

instvalid**<=**`InstValid**;**

immenable**<=** 1'b0**;**

**end**

`EXE\_OR**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_OR\_OP**;** //运算子类型是或运算

alusel\_o **<=**`EXE\_RES\_LOGIC**;** //运算类型是逻辑运算4

reg1\_read\_o **<=** 1'b1**;**

reg2\_read\_o **<=** 1'b1**;**

// imm <= {16'h0,inst\_i[15:0]};

wd\_o **<=** op1**;** //[7:4]为目的寄存器

instvalid **<=** `InstValid**;**

immenable **<=** 1'b0**;**

**end**

`EXE\_NOT**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_NOT\_OP**;**

alusel\_o **<=**`EXE\_RES\_LOGIC**;**

reg1\_read\_o **<=**1'b1**;**

reg2\_read\_o **<=**1'b0**;**

imm **<=**16'h0**;**

wd\_o **<=**op1**;**

instvalid**<=**`InstValid**;**

immenable **<=**1'b0**;**

**end**

`EXE\_XOR**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_XOR\_OP**;**

alusel\_o **<=**`EXE\_RES\_LOGIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg2\_read\_o **<=** 1'b1**;**

instvalid**<=**`InstValid**;**

immenable **<=**1'b0**;**

**end**

`EXE\_TEST**:** **begin**

wreg\_o **<=**`WriteDisable**;**

aluop\_o **<=**`EXE\_TEST\_OP**;**

alusel\_o **<=**`EXE\_RES\_LOGIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg2\_read\_o **<=** 1'b1**;**

instvalid**<=**`InstValid**;**

flag\_upd\_o **<=** 1'b1**;**

immenable **<=**1'b0**;**

**end**

`EXE\_SHL**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_SHL\_OP**;**

alusel\_o **<=**`EXE\_RES\_SHIFT**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b0**;**

flag\_upd\_o **<=**1'b1**;**

wd\_o **<=**op1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_SHR**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_SHR\_OP**;**

alusel\_o **<=**`EXE\_RES\_SHIFT**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b0**;**

flag\_upd\_o **<=**1'b1**;**

wd\_o **<=**op1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_SAR**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_SAR\_OP**;**

alusel\_o **<=**`EXE\_RES\_SHIFT**;**

reg1\_read\_o **<=**1'b1**;**

reg2\_read\_o **<=**1'b0**;**

wd\_o **<=**op1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

**default:** **begin**

**end**

**endcase**

**end**

`INST\_2**:begin** //立即数

**case(**op**)**

`EXE\_LOADH**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_ADD\_OP**;**

alusel\_o **<=**`EXE\_RES\_ARITHMETIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg1\_addr\_o **<=** 4'b1111**;**

reg2\_read\_o **<=** 1'b0**;**

imm **<=** **{**op1**,**op2**,**reg1\_data\_i**[**7**:**0**]};**

wd\_o **<=**4'b1111**;** //15号寄存器

instvalid **<=** `InstValid**;**

immenable **<=** 1'b1**;**

**end**

`EXE\_LOADL**:** **begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_ADD\_OP**;**

alusel\_o **<=**`EXE\_RES\_ARITHMETIC**;**

reg1\_read\_o **<=** 1'b1**;**

reg1\_addr\_o **<=** 4'b1111**;**

reg2\_read\_o **<=** 1'b0**;**

imm **<={**reg1\_data\_i**[**15**:**8**],**op1**,**op2**};**

wd\_o **<=**4'b1111**;**

instvalid **<=**`InstValid**;**

immenable **<=**1'b1**;**

**end**

**default:** **begin**

**end**

**endcase**

**end**

`INST\_3**:begin**

**case(**op**)**

`EXE\_LOAD**:** **begin**

wreg\_o**<=**`WriteEnable**;**

aluop\_o**<=**`EXE\_LOAD\_OP**;**

alusel\_o**<=**`EXE\_RES\_LOAD\_STORE**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

wd\_o **<=** op1**;**

instvalid**<=** `InstValid**;**

**end**

`EXE\_STORE**:** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_STORE\_OP**;**

alusel\_o**<=**`EXE\_RES\_LOAD\_STORE**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

**end**

**default:** **begin** **end**

**endcase**

**end**

`INST\_4**:begin** //NOP指令

**case(**op**)**

`EXE\_NOP**:** **begin**

wreg\_o **<=**`WriteDisable**;**

aluop\_o **<=**`EXE\_NOP\_OP**;**

alusel\_o **<=**`EXE\_RES\_NOP**;**

reg1\_read\_o**<=**1'b0**;**

reg2\_read\_o**<=**1'b0**;**

immenable**<=**1'b0**;**

instvalid**<=**`InstValid**;**

**end**

**default:** **begin**

**end**

**endcase**

**end**

`INST\_5**:begin** //移位指令 HILO

**case(**op**)**

`EXE\_MFHI**:begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_MFHI\_OP**;**

alusel\_o **<=**`EXE\_RES\_MOVE**;**

reg1\_read\_o**<=**1'b0**;**

reg2\_read\_o**<=**1'b0**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_MFLO**:begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_MFLO\_OP**;**

alusel\_o **<=**`EXE\_RES\_MOVE**;**

reg1\_read\_o**<=**1'b0**;**

reg2\_read\_o**<=**1'b0**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_MTHI**:begin**

wreg\_o **<=**`WriteDisable**;**

aluop\_o **<=**`EXE\_MTHI\_OP**;**

reg1\_read\_o**<=**1'b0**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_MTLO**:begin** //源寄存器到LO

wreg\_o **<=**`WriteDisable**;**

aluop\_o **<=**`EXE\_MTLO\_OP**;**

reg1\_read\_o**<=**1'b0**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_MOVE**:begin**

wreg\_o **<=**`WriteEnable**;**

aluop\_o **<=**`EXE\_MOVE\_OP**;**

alusel\_o **<=**`EXE\_RES\_MOVE**;**

reg1\_read\_o**<=**1'b0**;**

reg2\_read\_o**<=**1'b1**;**

wd\_o **<=**op1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_MOVN**:begin**

aluop\_o **<=**`EXE\_MOVN\_OP**;**

alusel\_o **<=**`EXE\_RES\_MOVE**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

**if(**reg1\_o**!=**`ZeroHalfWord**)** **begin**

wreg\_o**<=**`WriteEnable**;**

**end**

**else** **begin**

wreg\_o**<=**`WriteDisable**;**

**end**

**end**

`EXE\_MOVZ**:begin**

aluop\_o **<=**`EXE\_MOVN\_OP**;**

alusel\_o **<=**`EXE\_RES\_MOVE**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

**if(**reg1\_o**==**`ZeroHalfWord**)** **begin** //reg1为目的寄存器

wreg\_o**<=**`WriteEnable**;**

**end**

**else** **begin**

wreg\_o**<=**`WriteDisable**;**

**end**

**end**

**default:** **begin** **end**

**endcase**

**end**

`INST\_6**:begin** //乘除

**case(**op**)**

`EXE\_MUL**:** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_MUL\_OP**;**

alusel\_o**<=**`EXE\_RES\_MUL**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

flag\_upd\_o**<=**1'b1**;**

immenable**<=**1'b0**;**

**end**

`EXE\_MULU**:** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_MULU\_OP**;**

alusel\_o**<=**`EXE\_RES\_MUL**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_DIV**:** **begin** //aluopsel\_o 默认EXE\_RES\_NOP;

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_DIV\_OP**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

`EXE\_DIV**:** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_DIVU\_OP**;**

reg1\_read\_o**<=**1'b1**;**

reg2\_read\_o**<=**1'b1**;**

instvalid**<=**`InstValid**;**

immenable**<=**1'b0**;**

**end**

**default:** **begin** **end**

**endcase**

**end**

`INST\_7**:begin**

**case(**op**)**

`EXE\_J**:** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_J\_OP**;**

alusel\_o**<=**`EXE\_RES\_JUMP\_BRANCH**;**

reg1\_read\_o **<=**1'b0**;**

reg2\_read\_o **<=**1'b0**;**

immenable **<=**1'b0**;**

link\_addr\_o **<=**`ZeroHalfWord**;**

branch\_flag\_o **<=**`Branch**;**

next\_inst\_in\_delayslot\_o**<=**`InDelaySlot**;**

instvalid **<=** `InstValid**;**

branch\_target\_address\_o **<=** pc\_i **+{**8'h00**,**inst\_i**[**7**:**0**]};**

**end**

`EXE\_JC**:** **begin**

**if(**id\_c**==**1'b1**)** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_J\_OP**;**

alusel\_o**<=**`EXE\_RES\_JUMP\_BRANCH**;**

reg1\_read\_o **<=**1'b0**;**

reg2\_read\_o **<=**1'b0**;**

imm **<={**op1**,**op2**};**

link\_addr\_o **<=**`ZeroHalfWord**;**

branch\_flag\_o **<=**`Branch**;**

next\_inst\_in\_delayslot\_o**<=**`InDelaySlot**;**

instvalid **<=** `InstValid**;**

branch\_target\_address\_o **<=** pc\_i **+{**8'h00**,**pc\_i**[**7**:**0**]};**

**end**

**else** **begin** **end**

**end**

`EXE\_JNC**:** **begin**

**if(**id\_c**==**1'b0**)** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_J\_OP**;**

alusel\_o**<=**`EXE\_RES\_JUMP\_BRANCH**;**

reg1\_read\_o **<=**1'b0**;**

reg2\_read\_o **<=**1'b0**;**

immenable **<=**1'b0**;**

imm **<={**op1**,**op2**};**

link\_addr\_o **<=**`ZeroHalfWord**;**

branch\_flag\_o **<=**`Branch**;**

next\_inst\_in\_delayslot\_o**<=**`InDelaySlot**;**

instvalid **<=** `InstValid**;**

branch\_target\_address\_o **<=** pc\_i **+{**8'h00**,**inst\_i**[**7**:**0**]};**

**end**

**else** **begin** **end**

**end**

`EXE\_JZ**:** **begin**

**if(**id\_z**==**1'b1**)** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_J\_OP**;**

alusel\_o**<=**`EXE\_RES\_JUMP\_BRANCH**;**

reg1\_read\_o **<=**1'b0**;**

reg2\_read\_o **<=**1'b0**;**

immenable **<=**1'b0**;**

imm **<={**op1**,**op2**};**

link\_addr\_o **<=**`ZeroHalfWord**;**

branch\_flag\_o **<=**`Branch**;**

next\_inst\_in\_delayslot\_o**<=**`InDelaySlot**;**

instvalid **<=** `InstValid**;**

branch\_target\_address\_o **<=** pc\_i **+{**8'h00**,**inst\_i**[**7**:**0**]};**

**end**

**else** **begin** **end**

**end**

`EXE\_JNZ**:** **begin**

**if(**id\_z**==**1'b0**)** **begin**

wreg\_o**<=**`WriteDisable**;**

aluop\_o**<=**`EXE\_J\_OP**;**

alusel\_o**<=**`EXE\_RES\_JUMP\_BRANCH**;**

reg1\_read\_o **<=**1'b0**;**

reg2\_read\_o **<=**1'b0**;**

immenable **<=**1'b0**;**

imm **<={**op1**,**op2**};**

link\_addr\_o **<=**`ZeroHalfWord**;**

branch\_flag\_o **<=**`Branch**;**

next\_inst\_in\_delayslot\_o**<=**`InDelaySlot**;**

instvalid **<=** `InstValid**;**

branch\_target\_address\_o **<=** pc\_i **+{**8'h00**,**inst\_i**[**7**:**0**]};**

**end**

**else** **begin** **end**

**end**

**default:** **begin** **end**

**endcase**

**end**

**default:** **begin**

**end**

**endcase**

**end**

**end**

//延迟槽

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

is\_in\_delayslot\_o**<=**`NotInDelaySlot**;** //\_i当前译码指令是否是延迟槽指令

**end**

**else** **begin**

is\_in\_delayslot\_o **<=** is\_in\_delayslot\_i**;**

**end**

**end**

/\* 确定源操作数1 \*/

**always** **@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

reg1\_o**<=**`ZeroHalfWord**;**

**end**

**else** **if((**reg1\_read\_o**==**1'b1**)&&(**ex\_wreg\_i**==**1'b1**)&&(**reg1\_addr\_o**==**ex\_wd\_i**))** **begin** //译码阶段要读的寄存器是执行阶段要写的寄存器则

reg1\_o**<=**ex\_wdata**;** **end**

**else** **if((**reg1\_read\_o**==**1'b1**)&&(**mem\_wreg\_i**==**1'b1**)&&(**reg1\_addr\_o**==**mem\_wd\_i**))** **begin** //译码阶段要读的寄存器是访存阶段要写的寄存器

reg1\_o**<=**mem\_wdata**;** **end**

**else** **if((**reg1\_read\_o **==** 1'b1**)&&((**op**!=**`EXE\_LOADH**)||(**op**!=**`EXE\_LOADL**)))** **begin**

reg1\_o**<=**reg1\_data\_i**;** //寄存器组 读端口1 的输出值

**end**

// else if((reg1\_read\_o == 1'b0)&&(immenable ==1'b1)) begin

// reg1\_o<=imm; //立即数 如果立即数不参与运算，则在译码阶段将立即数置零

// end

**else** **begin**

reg1\_o**<=**`ZeroHalfWord**;**

**end**

**end**

/\* 确定源操作数2 \*/

**always** **@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

reg2\_o**<=**`ZeroHalfWord**;**

**end**

**else** **if((**reg2\_read\_o**==**1'b1**)&&(**ex\_wreg\_i**==**1'b1**)&&(**reg2\_addr\_o**==**ex\_wd\_i**))** **begin** //译码阶段要读的寄存器是执行阶段要写的寄存器则

reg2\_o**<=**ex\_wdata**;** **end**

**else** **if((**reg2\_read\_o**==**1'b1**)&&(**mem\_wreg\_i**==**1'b1**)&&(**reg2\_addr\_o**==**mem\_wd\_i**))** **begin** //译码阶段要读的寄存器是访存阶段要写的寄存器

reg2\_o**<=**mem\_wdata**;** **end**

**else** **if(**reg2\_read\_o **==** 1'b1**)** **begin**

reg2\_o**<=**reg2\_data\_i**;** //寄存器组 读端口2 的输出值

**end**

**else** **if((**reg2\_read\_o **==**1'b0**)&&(**immenable**==**1'b1**))** **begin**

reg2\_o**<=**imm**;**

**end**

**else** **begin**

reg2\_o**<=**`ZeroHalfWord**;**

**end**

**end**

**endmodule**

5、id\_ex.v

`include"define.v"

**module** id\_ex**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire[**5**:**0**]** stall**,**

//前一阶段（译码阶段）传递过来的信息

**input** **wire[**`AluOpBus**]** id\_aluop**,**

**input** **wire[**`AluSelBus**]** id\_alusel**,**

**input** **wire[**`RegBus**]** id\_reg1**,**

**input** **wire[**`RegBus**]** id\_reg2**,**

**input** **wire[**`RegAddrBus**]** id\_wd**,** //要写入的寄存器的地址

**input** **wire** id\_wreg**,** //译码阶段的指令是否有要写入的目的寄存器

**input** **wire** id\_flag\_upd**,**

**input** **wire** id\_is\_in\_delayslot**,** //当前处于译码阶段的指令是否位于延迟槽

**input** **wire[**`RegBus**]** id\_link\_address**,** //处于译码阶段的转移指令要保存的返回地址

**input** **wire** next\_inst\_in\_delayslot\_i**,**//下一条进入译码阶段的指令是否位于延迟槽

**output** **reg** ex\_is\_in\_delayslot**,** //当前处于执行阶段的指令是否位于延迟槽

**output** **reg[**`RegBus**]** ex\_link\_address**,** //处于执行阶段的转移指令要保存的返回地址

**output** **reg** is\_in\_delayslot\_o**,** //当前位于译码阶段的指令是否位于延迟槽

//传递到执行阶段的信息

**output** **reg[**`AluOpBus**]** ex\_aluop**,**

**output** **reg[**`AluSelBus**]** ex\_alusel**,**

**output** **reg[**`RegBus**]** ex\_reg1**,**

**output** **reg[**`RegBus**]** ex\_reg2**,**

**output** **reg[**`RegAddrBus**]** ex\_wd**,**

**output** **reg** ex\_wreg**,**

**output** **reg** ex\_flag\_upd

**);**

// 当stall[2]为stop stall[3] Nostop 表示译码暂停

// 执行阶段继续，所以使用空指令nop进入下一周期

// stall[2] nostop时，译码继续，指令进入执行阶段

// 其余阶段保持ex\_inst不变

**always** **@(posedge** clk**)** **begin**

**if(**rst **==** `RstEnable**)** **begin**

ex\_aluop **<=** `EXE\_NOP\_OP**;**

ex\_alusel**<=** `EXE\_RES\_NOP**;**

ex\_reg1 **<=** `ZeroHalfWord**;**

ex\_reg2 **<=** `ZeroHalfWord**;**

ex\_wd **<=** `NOPRegAddr**;**

ex\_wreg **<=** `WriteDisable**;**

ex\_flag\_upd**<=** `NoFlagUpd**;**

**end**

**else** **if(**stall**[**2**]==**`Stop**&&**stall**[**3**]==**`NoStop**)begin**

ex\_aluop **<=**`EXE\_NOP\_OP**;**

ex\_alusel**<=**`EXE\_RES\_NOP**;**

ex\_reg1 **<=**`ZeroHalfWord**;**

ex\_reg2 **<=**`ZeroHalfWord**;**

ex\_wd **<=**`NOPRegAddr**;**

ex\_wreg **<=**`WriteDisable**;**

**end**

**else** **if(**stall**[**2**]==**`NoStop**)begin**

ex\_aluop **<=** id\_aluop**;**

ex\_alusel**<=** id\_alusel**;**

ex\_reg1 **<=** id\_reg1**;**

ex\_reg2 **<=** id\_reg2**;**

ex\_wd **<=** id\_wd**;**

ex\_wreg **<=** id\_wreg**;**

ex\_flag\_upd**<=**id\_flag\_upd**;**

ex\_link\_address **<=** id\_link\_address**;**

ex\_is\_in\_delayslot**<=**id\_is\_in\_delayslot**;**

is\_in\_delayslot\_o**<=** next\_inst\_in\_delayslot\_i**;** //当前处于译码阶段的指令是否处于译码阶段

**end**

**end**

**endmodule**

6、ex.v

`include"define.v"

**module** ex**(**

**input** **wire** rst**,**

// 译码阶段传输到执行阶段的信息

**input** **wire[**`AluOpBus**]** aluop\_i**,**

**input** **wire[**`AluSelBus**]** alusel\_i**,**

**input** **wire[**`RegBus**]** reg1\_i**,**

**input** **wire[**`RegBus**]** reg2\_i**,**

**input** **wire[**`RegAddrBus**]** wd\_i**,**

**input** **wire** wreg\_i**,**

// 输入的标志位

**input** **wire** c\_i**,**

**input** **wire** v\_i**,**

**input** **wire** z\_i**,**

**input** **wire** s\_i**,**

**input** **wire** ex\_flag\_upd\_i**,**

// 输出的标志位

**output** **reg** c\_o**,**

**output** **reg** v\_o**,**

**output** **reg** z\_o**,**

**output** **reg** s\_o**,**

//执行的结果

**output** **reg[**`RegAddrBus**]** wd\_o**,**

**output** **reg** wreg\_o**,**

**output** **reg[**`RegBus**]** wdata\_o**,**

//HILO的输入值

**input** **wire[**`RegBus**]** hi\_i**,**

**input** **wire[**`RegBus**]** lo\_i**,**

//回写阶段指令是否要写HI,LO,用来检测hI,lo带来的数据相关问题

**input** **wire[**`RegBus**]** wb\_hi\_i**,**

**input** **wire[**`RegBus**]** wb\_lo\_i**,**

**input** **wire** wb\_whilo\_i**,**

//访存阶段的指令是否要写入HI,LO，用于检测数据相关

**input** **wire[**`RegBus**]** mem\_hi\_i**,**

**input** **wire[**`RegBus**]** mem\_lo\_i**,**

**input** **wire** mem\_whilo\_i**,**

//执行阶段的指令对HI，lo的写操作请求

**output** **reg** whilo\_o**,**

**output** **reg[**`RegBus**]** hi\_o**,**

**output** **reg[**`RegBus**]** lo\_o**,**

//除法指令

**input** **wire[**`DoubleRegBus**]** div\_result\_i**,**

**input** **wire** div\_ready\_i**,**

**output** **reg[**`RegBus**]** div\_opdata1\_o**,**

**output** **reg[**`RegBus**]** div\_opdata2\_o**,**

**output** **reg** div\_start\_o**,**

**output** **reg** signed\_div\_o**,**

**output** **reg** stallreq **,**

**input** **wire[**`RegBus**]** link\_address\_i**,** //处于执行阶段的指令要保存的返回地址

**input** **wire** is\_in\_delayslot\_i**,**//当前执行的指令是否位于延迟槽 异常处理时需要

// output reg[1:0] wr\_Mem,

**output** **reg[**`AluOpBus**]** aluop\_o**,**

**output** **reg[**`RegBus**]** mem\_addr\_o**,**

**output** **reg[**`RegBus**]** data\_store

**);**

**always** **@(\*)**

**begin**

aluop\_o **=**aluop\_i**;**

**end**

//always@(\*) begin

// if(rst==`RstEnable) begin wr\_Mem <= 2'b1x; end

// else begin

// wr\_Mem <= 2'b10;

// if(aluop\_i == `EXE\_LOAD\_OP) begin wr\_Mem <= 2'b00; end

// else if(aluop\_i ==`EXE\_STORE\_OP) begin wr\_Mem <= 2'b01; end

// end

// end

//assign aluop\_o =aluop\_i;

//assign mem\_addr\_o = (aluop\_i==`EXE\_LOAD\_OP)? reg2\_i:

// (aluop\_i==`EXE\_STORE\_OP)? reg1\_i:16'hzzzz;

//assign data\_store = (aluop\_i==`EXE\_STORE\_OP)? reg2\_i:16'hzzzz;

**always@(\*)** **begin**

**if(**rst **==**`RstEnable**)** **begin**

mem\_addr\_o **<=** 16'h0000**;**

**end**

**else** **begin**

mem\_addr\_o **<=** 16'h0000**;**

**if(**aluop\_i**==**`EXE\_LOAD\_OP**)** **begin**

mem\_addr\_o **<=** reg2\_i**;**

**end**

**else** **if(**aluop\_i**==**`EXE\_STORE\_OP**)** **begin**

mem\_addr\_o **<=** reg1\_i**;**

data\_store **<=** reg2\_i**;**

**end**

**end**

**end**

**reg** stallreq\_for\_div**;** //判断是否由于除法运算导致流水线暂停

**wire** z\_temp**;**

**reg** c\_temp\_1**;**

**wire** c\_temp**;**

**wire** s\_temp**;**

**wire** ov\_sum**;** //保存溢出情况

**wire** reg1\_eq\_reg2**;**

**wire** reg1\_lt\_reg2**;**

**reg[**`RegBus**]** arithmeticres**;**//保存算术运算的结果

**wire[**`RegBus**]** reg2\_i\_mux**;**//保存第二个操作数的补码

**wire[**`RegBus**]** reg1\_i\_not**;** //保存第一个操作数的反码

**wire[**`RegBus**]** result\_sum**;**//保存加法结果

**wire[**`RegBus**]** opdata1\_mult**;** //保存乘法中的被乘数

**wire[**`RegBus**]** opdata2\_mult**;** //乘法操作中的乘数

//保存逻辑运算的结果

**reg[**`RegBus**]** logicout**;**

//保存移位运算结果

**reg[**`RegBus**]** shiftres**;**

**reg[**`RegBus**]** moveres**;**

**reg[**`RegBus**]** HI**;**

**reg[**`RegBus**]** LO**;** //保存LO寄存器的最新值

**wire[**`DoubleRegBus**]** hilo\_temp**;** //临时保存乘法结果

**reg[**`DoubleRegBus**]** mulres**;** //保存乘法结果

//如果是减法或有符号运算 reg2\_i\_mux等于第二个操作数的补码

**assign** reg2\_i\_mux **=** **((**aluop\_i **==**`EXE\_SUB\_OP**)** **||**

**(**aluop\_i**==**`EXE\_CMP\_OP**))** **?**

**(~**reg2\_i**+**1**):**reg2\_i**;**

**assign** result\_sum **=** reg1\_i **+** reg2\_i\_mux**;**

**assign** z\_temp **=** **(((**aluop\_i**==**`EXE\_TEST\_OP**)&&(**logicout**==**16'b0**))||**

**((**aluop\_i**==**`EXE\_SUB\_OP**||**aluop\_i**==**`EXE\_ADD\_OP**||**aluop\_i**==**`EXE\_CMP\_OP**)&&(**result\_sum**==**16'b0**))**

**)?**1'b1**:**1'b0**;**

//计算是否溢出

**assign** ov\_sum **=((!**reg1\_i**[**15**]** **&&** **!**reg2\_i**[**15**])** **&&** result\_sum**[**15**])**

**||** **((**reg1\_i**[**15**]** **&&**reg2\_i**[**15**]** **&&(!**result\_sum**[**15**])));**

**assign** c\_temp **=(**aluop\_i**==**`EXE\_ADD\_OP**)?**ov\_sum**:~**ov\_sum**;** //加减法的c，v相同。

**assign** s\_temp **=** **(**aluop\_i **==**`EXE\_MUL\_OP**)** **?** mulres**[**31**]:** result\_sum**[**15**];** //符号运算

//符号位与最高位相同 //add sub dec inc cmp

**always@(\*)**

**begin**

**if(**rst**==**`RstEnable**)** **begin**

c\_o**<=**1'b0**;**

v\_o**<=**1'b0**;**

z\_o**<=**1'b0**;**

s\_o**<=**1'b0**;**

**end**

**else** **if(**ex\_flag\_upd\_i**==**1'b1**)** **begin**

c\_o**<=**c\_i**;**

v\_o**<=**v\_i**;**

z\_o**<=**z\_i**;**

s\_o**<=**s\_i**;**

**case(**aluop\_i**)**

`EXE\_ADD\_OP**:** **begin**

c\_o**<=**c\_temp**;**

v\_o**<=**ov\_sum**;**

**end**

`EXE\_SUB\_OP**:** **begin**

c\_o**<=**ov\_sum**;**

v\_o**<=**ov\_sum**;**

z\_o**<=**z\_temp**;**

**end**

`EXE\_TEST\_OP**,**`EXE\_CMP\_OP**:**

**begin** z\_o**<=**z\_temp**;** **end**

`EXE\_SHL\_OP**,**`EXE\_SHR\_OP**,**`EXE\_SAR\_OP**:begin**

c\_o**<=**c\_temp\_1**;**

**end**

`EXE\_MUL**:** **begin**

s\_o**<=**s\_temp**;**

**end**

**default:** **begin** **end**

**endcase**

**end**

// if((aluop\_i==`EXE\_SHL\_OP)||(aluop\_i==`EXE\_SHR\_OP))begin

// c\_o<=c\_temp\_1;

// end

// else begin c\_o<=c\_temp;end

// if((aluop\_i==`EXE\_ADD\_OP)||(aluop\_i==`EXE\_SUB\_OP))begin

// v\_o<=ov\_sum;

// end

// z\_o<=z\_temp;

// if((aluop\_i==`EXE\_ADD\_OP)||(aluop\_i==`EXE\_SUB\_OP)||(aluop\_i==`EXE\_MUL\_OP))

// s\_o<=s\_temp;

// end

**end**

//计算操作数是否小于操作二

//as

//解决HI LO的数据相关

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

**{**HI**,**LO**}<={**`ZeroHalfWord**,**`ZeroHalfWord**};**

**end** **else** **if(**mem\_whilo\_i**==**`WriteEnable**)** **begin**

**{**HI**,**LO**}<={**mem\_hi\_i**,**mem\_lo\_i**};**// 访存阶段的指令前推

**end** **else** **if(**wb\_whilo\_i **==**`WriteEnable**)** **begin**

**{**HI**,**LO**}<={**wb\_hi\_i**,**wb\_lo\_i**};** //回写阶段的指令前推

**end** **else** **begin**

**{**HI**,**LO**}<={**hi\_i**,**lo\_i**};**

**end**

**end**

/\* 根据aluop\_i指示的运算子类型进行运算， 逻辑或 运算 \*/

**always** **@(\*)** **begin**

**if(**rst **==** `RstEnable**)** **begin** logicout**<=**`ZeroHalfWord**;** **end**

**else** **begin**

**case(**aluop\_i**)**

`EXE\_OR\_OP**:** **begin** logicout **<=** reg1\_i **|** reg2\_i**;** **end**

`EXE\_AND\_OP**:begin** logicout **<=** reg1\_i **&** reg2\_i**;** **end**

`EXE\_NOT\_OP**:begin** logicout **<=** **~**reg1\_i**;** **end**

`EXE\_XOR\_OP**:begin** logicout **<=** reg1\_i **^** reg2\_i**;** **end**

`EXE\_TEST\_OP**:begin** logicout**<=** reg1\_i **&** reg2\_i**;** **end**

**default:** **begin** logicout **<=** `ZeroHalfWord**;** **end**

**endcase**

**end**

**end**

//算术运算

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

arithmeticres **<=** `ZeroHalfWord**;** **end**

**else** **begin**

**case(**aluop\_i**)**

`EXE\_ADD\_OP**:** **begin** arithmeticres**<=**result\_sum**;** **end**

`EXE\_SUB\_OP**:** **begin** arithmeticres**<=**result\_sum**;** **end**

**default:** **begin** arithmeticres**<=**`ZeroHalfWord**;** **end**

**endcase**

**end**

**end**

//乘法运算

**assign** opdata1\_mult **=((**aluop\_i**==**`EXE\_MUL\_OP**)&&(**reg1\_i**[**15**]==**1'b1**))?(~**reg1\_i**+**1**):**reg1\_i**;**

**assign** opdata2\_mult **=((**aluop\_i**==**`EXE\_MUL\_OP**)&&(**reg2\_i**[**15**]==**1'b1**))?(~**reg2\_i**+**1**):**reg2\_i**;**

**assign** hilo\_temp **=** opdata1\_mult**\***opdata2\_mult**;** //保存临时乘法的结果

//乘法的临时结果需要修正，

//1.有符号乘法， 如果被乘数与乘数一正一负，需对结果求补码

//2.有符号 两乘数同正负，则无需修正，无符号无需修正

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

mulres**<={**`ZeroHalfWord**,**`ZeroHalfWord**};**

**end** **else** **if(**aluop\_i**==**`EXE\_MUL\_OP**)**

**begin**

**if(**reg1\_i**[**15**]^**reg2\_i**[**15**]==**1'b1**)** **begin**

mulres**<=** **~**hilo\_temp **+** 1**;**

**end** **else** **begin**

mulres**<=** hilo\_temp**;**

**end**

**end** **else** **begin**

mulres**<=** hilo\_temp**;**

**end**

**end**

//除法运算 输出除法器控制信息，获取触发器的结果信号

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

stallreq\_for\_div **<=** `NoStop**;**

div\_opdata1\_o **<=** `ZeroHalfWord**;**

div\_opdata2\_o **<=** `ZeroHalfWord**;**

div\_start\_o **<=** `DivStop**;**

signed\_div\_o **<=** 1'b0**;**

**end**

**else** **begin**

stallreq\_for\_div **<=** `NoStop**;**

div\_opdata1\_o **<=** `ZeroHalfWord**;**

div\_opdata2\_o **<=** `ZeroHalfWord**;**

div\_start\_o **<=** `DivStop**;**

signed\_div\_o **<=** 1'b0**;**

**case(**aluop\_i**)**

`EXE\_DIV\_OP**:** **begin**

**if(**div\_ready\_i **==** `DivResultNotReady**)** **begin**

div\_opdata1\_o **<=** reg1\_i**;**

div\_opdata2\_o **<=** reg2\_i**;**

div\_start\_o **<=** `DivStart**;**

signed\_div\_o **<=** 1'b1**;**

stallreq\_for\_div**<=**`Stop**;**

**end**

**else** **if(**div\_ready\_i **==** `DivResultReady**)** **begin**

div\_opdata1\_o **<=** reg1\_i**;**

div\_opdata2\_o **<=** reg2\_i**;**

div\_start\_o **<=** `DivStop**;**

signed\_div\_o **<=** 1'b1**;**

stallreq\_for\_div**<=**`NoStop**;**

**end**

**else** **begin**

div\_opdata1\_o **<=** `ZeroHalfWord**;**

div\_opdata2\_o **<=** `ZeroHalfWord**;**

signed\_div\_o **<=** 1'b0**;**

div\_start\_o **<=** `DivStop**;**

stallreq\_for\_div**<=**`NoStop**;**

**end**

**end**

`EXE\_DIVU\_OP**:** **begin**

**if(**div\_ready\_i **==**`DivResultNotReady**)** **begin**

div\_opdata1\_o **<=** reg1\_i**;**

div\_opdata2\_o **<=** reg2\_i**;**

div\_start\_o **<=** `DivStart**;**

signed\_div\_o **<=** 1'b0**;**

stallreq\_for\_div**<=**`Stop**;**

**end**

**else** **if(**div\_ready\_i **==** `DivResultReady**)** **begin**

div\_opdata1\_o **<=** reg1\_i**;**

div\_opdata2\_o **<=** reg2\_i**;**

div\_start\_o **<=** `DivStop**;**

signed\_div\_o **<=** 1'b0**;**

stallreq\_for\_div**<=**`NoStop**;**

**end**

**else** **begin**

div\_opdata1\_o **<=** `ZeroHalfWord**;**

div\_opdata2\_o **<=** `ZeroHalfWord**;**

signed\_div\_o **<=** 1'b0**;**

div\_start\_o **<=** `DivStop**;**

stallreq\_for\_div**<=**`NoStop**;**

**end**

**end**

**default:** **begin** **end**

**endcase**

**end**

**end**

**always@(\*)** **begin**

stallreq **=** stallreq\_for\_div **||** **(**1'b0**);**

**end**

// 移位运算

**always@(\*)begin**

**if(**rst**==**`RstEnable**)** **begin** shiftres **<=**`ZeroHalfWord**;** **end**

**else** **begin**

**case(**aluop\_i**)**

`EXE\_SHL\_OP**:** **begin** shiftres **<=** **{**reg1\_i**[**14**:**0**],**1'b0**};**

// 待添加状态更新c

c\_temp\_1**<=** reg1\_i**[**15**];**

**end**

`EXE\_SHR\_OP**:** **begin** shiftres **<=** **{**1'b0**,**reg1\_i**[**15**:**1**]};**

c\_temp\_1**<=** reg1\_i**[**0**];**

**end**

`EXE\_SAR**:** **begin** shiftres **<=** **{**reg1\_i**[**0**],**reg1\_i**[**15**:**1**]};** **end**

**default:** **begin** shiftres **<=** `ZeroHalfWord**;end**

**endcase**

**end**

**end**

// 移动运算

**always@(\*)begin**

**if(**rst**==**`RstEnable**)** **begin** moveres**<=**`ZeroHalfWord**;** **end**

**else** **begin**

moveres**<=**`ZeroHalfWord**;**

**case(**aluop\_i**)**

`EXE\_MFHI\_OP**:** **begin** moveres**<=**HI**;** **end**

`EXE\_MFLO\_OP**:** **begin** moveres**<=**LO**;** **end**

`EXE\_MOVE\_OP**:** **begin** moveres**<=**reg2\_i**;end**

`EXE\_MOVZ\_OP**:** **begin** moveres**<=**reg2\_i**;end**

`EXE\_MOVN\_OP**:** **begin** moveres**<=**reg2\_i**;end**

**default:** **begin** **end**

**endcase**

**end**

**end**

//根据alusel\_i指示的运算类型，选择一个运算结果作为最终结果 \*/

**always@(\*)** **begin**

wd\_o **<=** wd\_i**;** //要写入的寄存器地址

wreg\_o **<=** wreg\_i**;** //是否要写入寄存器

**case(**alusel\_i**)**

`EXE\_RES\_LOGIC**:** **begin** wdata\_o**<=**logicout**;** **end**

`EXE\_RES\_SHIFT**:** **begin** wdata\_o**<=**shiftres**;** **end**

`EXE\_RES\_MOVE**:** **begin** wdata\_o**<=**moveres**;** **end**

`EXE\_RES\_ARITHMETIC**:** **begin** wdata\_o**<=**arithmeticres**;** **end**

`EXE\_RES\_MUL**:** **begin** wdata\_o **<=**mulres**[**15**:**0**];** **end**

`EXE\_RES\_JUMP\_BRANCH**:** **begin** //wdata\_o <=link\_address\_i;

**end**

**default:** **begin** wdata\_o**<=**`ZeroHalfWord**;** **end**

**endcase**

**end**

//向HI，Lo寄存器写值时，对控制信号的赋值

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

whilo\_o**<=**`WriteDisable**;**

hi\_o**<=**`ZeroHalfWord**;**

lo\_o**<=**`ZeroHalfWord**;**

**end**

**else** **if(**aluop\_i**==**`EXE\_MTHI\_OP**)** **begin**

whilo\_o**<=**`WriteEnable**;**

hi\_o**<=**reg2\_i**;**

lo\_o**<=**LO**;**

**end** **else** **if(**aluop\_i**==**`EXE\_MTLO\_OP**)** **begin**

whilo\_o**<=**`WriteEnable**;**

hi\_o**<=**HI**;**

lo\_o**<=**reg2\_i**;**

**end** **else** **if((**aluop\_i**==**`EXE\_MUL\_OP**)||**

**(**aluop\_i**==**`EXE\_MULU\_OP**))begin**

whilo\_o**<=**`WriteEnable**;**

hi\_o**<=**mulres**[**31**:**16**];**

lo\_o**<=**mulres**[**15**:**0**];**

**end** **else** **if((**aluop\_i**==**`EXE\_DIV\_OP**)||**

**(**aluop\_i**==**`EXE\_DIVU\_OP**))begin**

whilo\_o**<=**`WriteEnable**;**

hi\_o**<=**div\_result\_i**[**31**:**16**];** //余数

lo\_o**<=**div\_result\_i**[**15**:**0**];** //商

**end** **else** **begin**

whilo\_o**<=**`WriteDisable**;**

hi\_o**<=**`ZeroHalfWord**;**

lo\_o**<=**`ZeroHalfWord**;**

**end**

**end**

**endmodule**

7、ex\_mem.v

`include"define.v"

**module** ex\_mem**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire[**5**:**0**]** stall**,**

// 来自执行阶段的信息

**input** **wire[**`RegAddrBus**]** ex\_wd**,**

**input** **wire** ex\_wreg**,**

**input** **wire[**`RegBus**]** ex\_wdata**,**

**input** **wire[**`RegBus**]** ex\_hi**,**

**input** **wire[**`RegBus**]** ex\_lo**,**

**input** **wire** ex\_whilo**,**

**input** **wire[**`AluOpBus**]** ex\_aluop**,**

**input** **wire[**`RegBus**]** ex\_mem\_addr**,**

**input** **wire[**`RegBus**]** ex\_data\_store**,**

// 送到访存阶段的信息

**output** **reg[**`RegAddrBus**]** mem\_wd**,**

**output** **reg** mem\_wreg**,**

**output** **reg[**`RegBus**]** mem\_wdata**,**

**output** **reg[**`RegBus**]** mem\_hi**,**

**output** **reg[**`RegBus**]** mem\_lo**,**

**output** **reg** mem\_whilo**,**

**output** **reg[**`AluOpBus**]** mem\_aluop**,**

**output** **reg[**`RegBus**]** mem\_mem\_addr**,**

**output** **reg[**`RegBus**]** mem\_data\_store

**);**

// 当stall[3]为stop，stop[4] nostop时，表示执行阶段暂停

// 而访存阶段继续，所以采用空指令作为下一周期进入访存阶段的指令

// stall[3]为nostop ，执行阶段继续，执行后的指令进入访存阶段

// 其余情况，保持 输出信号不变

**always@(posedge** clk**)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

mem\_wd **<=** `NOPRegAddr**;**

mem\_wreg **<=** `WriteDisable**;**

mem\_wdata**<=** `ZeroHalfWord**;**

mem\_hi **<=**`ZeroHalfWord**;**

mem\_lo **<=**`ZeroHalfWord**;**

mem\_whilo**<=**`WriteDisable**;**

mem\_aluop**<=**`EXE\_NOP\_OP**;**

mem\_mem\_addr**<=**`ZeroHalfWord**;**

mem\_data\_store**<=**`ZeroHalfWord**;**

**end**

**else** **if(**stall**[**3**]==**`Stop**&&** stall**[**4**]==**`NoStop**)begin**

mem\_wd **<=**`NOPRegAddr**;**

mem\_wreg **<=**`WriteDisable**;**

mem\_wdata**<=**`ZeroHalfWord**;**

mem\_hi **<=**`ZeroHalfWord**;**

mem\_lo **<=**`ZeroHalfWord**;**

mem\_whilo**<=**`WriteDisable**;**

mem\_aluop**<=**`EXE\_NOP\_OP**;**

mem\_mem\_addr**<=**`ZeroHalfWord**;**

mem\_data\_store**<=**`ZeroHalfWord**;**

**end**

**else** **if(**stall**[**3**]==**`NoStop**)begin**

mem\_wd **<=** ex\_wd**;**

mem\_wreg **<=** ex\_wreg**;**

mem\_wdata**<=** ex\_wdata**;**

mem\_hi **<=** ex\_hi**;**

mem\_lo **<=** ex\_lo**;**

mem\_whilo**<=** ex\_whilo**;**

mem\_aluop**<=** ex\_aluop**;**

mem\_mem\_addr**<=** ex\_mem\_addr**;**

mem\_data\_store**<=** ex\_data\_store**;**

**end**

**end**

**endmodule**

8、mem.v

`include"define.v"

**module** mem**(**

**input** **wire** rst**,**

//来自执行阶段的信息

**input** **wire[**`RegAddrBus**]** wd\_i**,**

**input** **wire** wreg\_i**,**

**input** **wire[**`RegBus**]** wdata\_i**,**

**input** **wire[**`RegBus**]** hi\_i**,**

**input** **wire[**`RegBus**]** lo\_i**,**

**input** **wire** whilo\_i**,**

**input** **wire[**`AluOpBus**]** aluop\_i**,**

//

**input** **wire[**`RegBus**]** mem\_addr\_i**,**

**input** **wire[**`RegBus**]** mem\_data\_store**,**//需要store的数据

//load 的数据

**input** **wire[**`RegBus**]** mem\_data\_i**,**

//访存阶段的结果

**output** **reg[**`RegAddrBus**]** wd\_o**,**

**output** **reg** wreg\_o**,**

**output** **reg[**`RegBus**]** wdata\_o**,**

**output** **reg[**`RegBus**]** hi\_o**,**

**output** **reg[**`RegBus**]** lo\_o**,**

**output** **reg** whilo\_o**,**

///////////////////////////////////

**output** **reg[**`RegBus**]** mem\_addr\_o**,**

**output** **reg[**`RegBus**]** mem\_data\_o**,**//访存输出的数据

**output** **reg[**1**:**0**]** wr\_Mem

**);**

**always@(\*)** **begin**

**if(**rst **==** `RstEnable**)** **begin**

wd\_o **<=** `NOPRegAddr**;**

wreg\_o **<=** `WriteDisable**;**

wdata\_o **<=** `ZeroHalfWord**;**

hi\_o **<=** `ZeroHalfWord**;**

lo\_o **<=** `ZeroHalfWord**;**

whilo\_o **<=** `WriteDisable**;**

//ram

mem\_addr\_o **<=** `ZeroHalfWord**;**

mem\_data\_o **<=** `ZeroHalfWord**;**

wr\_Mem **<=** 2'b11**;**

**end**

**else** **begin**

wd\_o **<=** wd\_i**;**

wreg\_o **<=** wreg\_i**;**

wdata\_o **<=** wdata\_i**;**

hi\_o **<=** hi\_i**;**

lo\_o **<=** lo\_i**;**

whilo\_o **<=** whilo\_i**;**

mem\_data\_o**<=**`ZeroHalfWord**;**

mem\_addr\_o**<=**`ZeroHalfWord**;**

wr\_Mem **<=** 2'b11**;**

**case(**aluop\_i**)**

`EXE\_LOAD\_OP**:begin**

mem\_addr\_o **<=** mem\_addr\_o**;**

wr\_Mem **<=**2'b00**;**

wdata\_o **<=** mem\_data\_i**;**

**end**

`EXE\_STORE\_OP**:begin**

mem\_addr\_o **<=** mem\_addr\_i**;**

wr\_Mem **<=** 2'b01**;**

mem\_data\_o **<=** mem\_data\_store**;**

**end**

**endcase**

**end**

**end**

**endmodule**

9、mem\_wb.v

`include"define.v"

**module** mem\_wb**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire[**5**:**0**]** stall**,**

//访存阶段的结果

**input** **wire[**`RegAddrBus**]** mem\_wd**,**

**input** **wire** mem\_wreg**,**

**input** **wire[**`RegBus**]** mem\_wdata**,**

**input** **wire[**`RegBus**]** mem\_hi**,**

**input** **wire[**`RegBus**]** mem\_lo**,**

**input** **wire** mem\_whilo**,**

//送到回写阶段的信息

**output** **reg[**`RegAddrBus**]** wb\_wd**,**

**output** **reg** wb\_wreg**,**

**output** **reg[**`RegBus**]** wb\_wdata**,**

**output** **reg[**`RegBus**]** wb\_hi**,**

**output** **reg[**`RegBus**]** wb\_lo**,**

**output** **reg** wb\_whilo

**);**

//stall[4]为Stop，stall[5]为nostop时，表示访存暂停

// 回写阶段继续，用空指令作为下一周期进入回写阶段的指令

// srall[4]为nostop时，访存继续，访存后的指令进入回写阶段

// 其他情况，保持回写阶段的寄存器输出不变

**always@(posedge** clk**)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

wb\_wd **<=** `NOPRegAddr**;**

wb\_wreg **<=** `WriteDisable**;**

wb\_wdata **<=** `ZeroHalfWord**;**

wb\_hi **<=** `ZeroHalfWord**;**

wb\_lo **<=** `ZeroHalfWord**;**

wb\_whilo **<=** `WriteDisable**;**

**end**

**else** **if(**stall**[**4**]==**`Stop**&&**stall**[**5**]==**`NoStop**)** **begin**

wb\_wd **<=** `NOPRegAddr**;**

wb\_wreg **<=** `WriteDisable**;**

wb\_wdata **<=** `ZeroHalfWord**;**

wb\_hi **<=** `ZeroHalfWord**;**

wb\_lo **<=** `ZeroHalfWord**;**

wb\_whilo **<=** `WriteDisable**;**

**end**

**else** **if(**stall**[**4**]==**`NoStop**)begin**

wb\_wd **<=** mem\_wd**;**

wb\_wreg **<=** mem\_wreg**;**

wb\_wdata **<=** mem\_wdata**;**

wb\_hi **<=** mem\_hi**;**

wb\_lo **<=** mem\_lo**;**

wb\_whilo **<=** mem\_whilo**;**

**end**

**end**

**endmodule**

10、regfile.v

`include"define.v"

**module** regfile**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

// input wire[1:0] sel,

**input** **wire[**3**:**0**]** reg\_sel**,**

//xie

**input** **wire** we**,**

**input** **wire[**`RegAddrBus**]** waddr**,**

**input** **wire[**`RegBus**]** wdata**,**

// du 1

**input** **wire** re1**,**

**input** **wire[**`RegAddrBus**]** raddr1**,**

**output** **reg[**`RegBus**]** rdata1**,**

//du2

**input** **wire** re2**,**

**input** **wire[**`RegAddrBus**]** raddr2**,**

**output** **reg[**`RegBus**]** rdata2**,**

//

**output** **wire[**`RegBus**]** regfile\_data

**);**

**reg[**`RegBus**]** regs**[**`RegNum**-**1**:**0**];**

**always@(posedge** clk**)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

$readmemh**(**"regs\_reset.txt"**,**regs**);**

**end**

**else** **begin**

**if(**we**==**`WriteEnable**)** **begin**

regs**[**waddr**]** **<=** wdata**;**

**end**

**end**

**end**

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

rdata1**<=**`ZeroHalfWord**;**

**end**

**else** **if((**raddr1**==**waddr**)&&(**we**==**`WriteEnable**)&&(**re1**==**`ReadEnable**))** **begin**

rdata1**<=**wdata**;**

**end**

**else** **if(**re1**==**`ReadEnable**)** **begin**

rdata1**<=**regs**[**raddr1**];**

**end**

**else** **begin**

rdata1**<=**`ZeroHalfWord**;**

**end**

**end**

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

rdata2**<=**`ZeroHalfWord**;**

**end**

**else** **if((**raddr2**==**waddr**)&&(**we**==**`WriteEnable**)&&(**re2**==**`ReadEnable**))** **begin**

rdata2**<=**wdata**;**

**end**

**else** **if(**re2**==**`ReadEnable**)** **begin**

rdata2**<=**regs**[**raddr2**];**

**end**

**else** **begin**

rdata2**<=**`ZeroHalfWord**;**

**end**

**end**

//always@(\*) begin

// if((sel==2'b00)) begin

// reg\_data <= regs[reg\_sel];

// end

// else begin

// end

//end

**assign** regfile\_data **=** regs**[**reg\_sel**]** **;**

**endmodule**

11、divide.v

`include"define.v"

**module** divide**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire** signed\_div\_i**,**

**input** **wire[**15**:**0**]** opdata1\_i**,**

**input** **wire[**15**:**0**]** opdata2\_i**,**

**input** **wire** start\_i**,**

**input** **wire** annul\_i**,** //1时取消除法运算

**output** **reg[**31**:**0**]** result\_o**,**

**output** **reg** ready\_o //除法结束

**);**

**wire[**16**:**0**]** div\_temp**;**

**wire[**15**:**0**]** temp**;**

**reg[**4**:**0**]** cnt**;** //记录试商法进行次数，等于15则表示结束

**reg[**32**:**0**]** dividend**;**//低16位保存被除数的中间结果，第k次迭代结束，时[k:0]保存的为当前得到的中间结果，dividend[31:k+1]保存的是被除数中还没有参与运算的数据，

//dividend 高16位是每次迭代时的被减数，所以 [31:16]

**reg[**1**:**0**]** state**;**

**reg[**15**:**0**]** divisor**;** //除数

**reg[**15**:**0**]** temp\_op1**;**

**reg[**15**:**0**]** temp\_op2**;**

**assign** div\_temp **={**1'b0**,** dividend**[**31**:**16**]}-** **{**1'b0**,**divisor**}** **;**//- {1'b0,divisor}

**always@(posedge** clk**)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

state**<=**`DivFree**;**

ready\_o**<=**`DivResultNotReady**;**

result\_o**<={**`ZeroHalfWord**,**`ZeroHalfWord**};**

**end**

**else** **begin**

**case(**state**)**

`DivFree**:** **begin**

**if(**start\_i **==**`DivStart **&&**annul\_i**==**1'b0**)** **begin**

**if(**opdata2\_i **==**`ZeroHalfWord**)** **begin**

state**<=**`DivByZero**;**

**end**

**else** **begin**

state**<=**`DivOn**;**

cnt**<=**5'b00000**;**

**if(**signed\_div\_i **==**1'b1 **&&** opdata1\_i**[**15**]==**1'b1**)** **begin** //判断有无符号

temp\_op1 **=** **~**opdata1\_i **+**1**;** **end**

**else** **begin** temp\_op1 **=** opdata1\_i**;** **end**

**if(**signed\_div\_i **==**1'b1 **&&** opdata2\_i**[**15**]==**1'b1**)** **begin**

temp\_op2 **=** **~**opdata2\_i**+**1**;** **end**

**else** **begin** temp\_op2 **=** opdata2\_i**;** **end**

dividend **<=** **{**`ZeroHalfWord**,**`ZeroHalfWord**};**

dividend**[**16**:**1**]<=**temp\_op1**;**

divisor **<=** temp\_op2**;**

**end**

**end**

**else** **begin** //没有开始除法运算

ready\_o**<=**`DivResultNotReady**;**

result\_o**<={**`ZeroHalfWord**,**`ZeroHalfWord**};**

**end**

**end**

`DivByZero**:** **begin**

dividend**<={**`ZeroHalfWord**,**`ZeroHalfWord**};**

state**<=**`DivEnd**;**

**end**

`DivOn**:** **begin**

**if(**annul\_i **==**1'b0**)begin**

**if(**cnt**!=**5'b10000**)** **begin** //表示试商法还没结束

**if(**div\_temp**[**16**]==**1'b1**)** **begin**

// div\_temp ={1'b0, dividend[31:16]} - {1'b0,divisor};

//如果dic\_temp[16]为1，表示，minuend小于除数advisor,

//将dividend向左移1位，这样将被除数没有参与运算的最高位，加入到下一次

//迭代的被减数中，同时将0追加到中间结果。

dividend**<={**dividend**[**31**:**0**],**1'b0**};**

**end**

**else** **begin**

//如果div\_temp[16]=0,代表被减数大于等于除数

//将减法的结果与被除数没有参与运算的最高位加入到下一次迭代

//的被减数中，同时将1追加到中间结果

dividend**<={**div\_temp**[**15**:**0**],**dividend**[**15**:**0**],**1'b1**};**

**end**

cnt**<=**cnt**+**1**;**

**end**

**else** **begin** //试商法结束

**if((**signed\_div\_i**==**1'b1**)&&((**opdata1\_i**[**15**]^**opdata2\_i**[**15**])==**1'b1**))** **begin**//两数符号不同

dividend**[**15**:**0**]<=** **(~**dividend**[**15**:**0**]+**1**);** **end**

**if((**signed\_div\_i**==**1'b1**)&&((**opdata1\_i**[**15**]^**dividend**[**32**])==**1'b1**))begin**

dividend**[**32**:**17**]<=(~**dividend**[**32**:**17**]+**1**);** **end**

state**<=**`DivEnd**;**

cnt**<=**5'b00000**;**

**end**

**end** **else** **begin** //annul\_i为1

state**<=**`DivFree**;**

**end**

**end**

`DivEnd**:** **begin**

result\_o**<={**dividend**[**32**:**17**],**dividend**[**15**:**0**]};**

ready\_o **<=** `DivResultReady**;**

**if(**start\_i**==**`DivStop**)** **begin**

state**<=**`DivFree**;**

ready\_o**<=**`DivResultNotReady**;**

result\_o**<={**`ZeroHalfWord**,**`ZeroHalfWord**};**

**end**

**end**

**endcase**

**end**

**end**

**endmodule**

12、hilo\_reg.v

`include"define.v"

**module** hilo\_reg**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire** we**,**

**input** **wire[**`RegBus**]** hi\_i**,**

**input** **wire[**`RegBus**]** lo\_i**,**

//读端口

**output** **reg[**`RegBus**]** hi\_o**,**

**output** **reg[**`RegBus**]** lo\_o

**);**

**always@(posedge** clk**)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

hi\_o**<=**`ZeroHalfWord**;**

lo\_o**<=**`ZeroHalfWord**;**

**end**

**else** **if(**we**==**`WriteEnable**)** **begin**

hi\_o**<=**hi\_i**;**

lo\_o**<=**lo\_i**;**

**end**

**end**

**endmodule**

13、flag\_reg.v

`include"define.v"

**module** flag\_reg**(**

**input** **wire** clk**,**

**input** **wire** rst**,**

**input** **wire** c\_i**,**

**input** **wire** z\_i**,**

**input** **wire** v\_i**,**

**input** **wire** s\_i**,**

**output** **reg** c\_o**,**

**output** **reg** z\_o**,**

**output** **reg** v\_o**,**

**output** **reg** s\_o

**);**

**always@(posedge** clk**)** **begin**

c\_o**<=**1'b0**;**

z\_o**<=**1'b0**;**

v\_o**<=**1'b0**;**

s\_o**<=**1'b0**;**

**if(**rst**==**`RstEnable**)** **begin**

c\_o**<=**1'b0**;**

z\_o**<=**1'b0**;**

v\_o**<=**1'b0**;**

s\_o**<=**1'b0**;**

**end**

**else** **begin**

c\_o**<=**c\_i**;**

z\_o**<=**z\_i**;**

v\_o**<=**v\_i**;**

s\_o**<=**s\_i**;**

**end**

**end**

**endmodule**

14、ctrl.v

`include"define.v"

**module** ctrl**(**

**input** **wire** rst**,**

**input** **wire** stallreq\_from\_id**,**

**input** **wire** stallreq\_from\_ex**,**

**output** **reg[**5**:**0**]** stall

**);**

**always@(\*)** **begin**

**if(**rst**==**`RstEnable**)** **begin**

stall**<=**6'b000000**;**

**end**

**else** **if(**stallreq\_from\_ex**==**`Stop**)** **begin**

stall**<=**6'b001111**;**

**end**

**else** **if(**stallreq\_from\_id**==**`Stop**)** **begin**

stall**<=**6'b000111**;**

**end**

**else** **begin**

stall**<=**6'b000000**;**

**end**

**end**

**endmodule**

15、inst\_rom.v(仿真用)

`include"define.v"

**module** inst\_rom**(**

**input** **wire** we**,**

**input** **wire[**`InstAddrBus**]** addr**,**

**inout** **[**`InstBus**]** inst

**);**

**reg[**`RegBus**]** data\_in**;**

**reg[**`RegBus**]** data\_out**;**

//定义一个数组，大小为INStNUM，元素宽度为INnstBus

**reg[**`InstBus**]** inst\_mem**[**0**:**`InstNum**-**1**];**

//使用文件Inst)rom.data初始化指令存储器

**initial** $readmemh **(**"inst\_test1.txt"**,**inst\_mem**);**

//当复位信号无效时，依据输入的地址，输出指令存储器rom中的对应元素

**always@(\*)** **begin**

**if(**we**==**`ChipRead**)** **begin**

data\_out **<=**inst\_mem**[**addr**[**`InstMemNumLog2**+**1**:**1**]];**

**end**

**else** **begin**

inst\_mem**[**addr**[**`InstMemNumLog2**+**1**:**1**]]<=** data\_in**;**//一个地址对应两个字节数据，所以从第二位开始

**end**

**end**

//rom输出时

**assign** inst **=** **(**we**==**`ChipRead**)?** data\_out**:**16'hzzzz**;**

**always@(\*)**

**begin**

data\_in **<=**inst**;**

**end**

**endmodule**

16、top\_test.v（仿真）

`include"define.v"

**module** top\_test**(**

**input** clk**,**

**input** rst

**);**

**wire[**`InstAddrBus**]** inst\_addr**;**

**wire[**`InstBus**]** inst**;**

**wire** rom\_we**;**

**wire** c**,**z**,**v**,**s**;**

cpu\_try cpu\_try0**(**

**.**clk**(**clk**),** **.**rst**(**rst**),**

**.**ram\_addr\_o**(**inst\_addr**),** **.**ram\_data\_i**(**inst**),**

**.**ram\_we\_o**(**rom\_we**)** **,.**c**(**c**),** **.**z**(**z**),** **.**v**(**v**),** **.**s**(**s**)**

**);**

inst\_rom inst\_rom0**(**

**.**we**(**rom\_we**),** **.**addr**(**inst\_addr**),** **.**inst**(**inst**)**

**);**

**endmodule**

17、cpu\_try\_tb.v(testbench)

`timescale 1ns**/**1ps

`include"define.v"

**module** cpu\_try\_tb**();**

**reg** CLOCK\_50**;**

**reg** rst**;**

//对寄存器组初始化

//每隔10ns clk信号翻转一次，一个周期20ns 对应50mhz

**initial** **begin**

CLOCK\_50 **=** 1'b0**;**

**forever** **#**50 CLOCK\_50 **=** **~**CLOCK\_50**;**

**end**

**initial** **begin**

rst **=** `RstEnable**;**

**#**195 rst **=** `RstDisable**;**

**#**60000 $stop**;**

**end**

//最初时刻，复位信号有效，在195ns，复位信号无效，最小SOPC运行

top\_test top\_test0**(**

**.**clk**(**CLOCK\_50**),**

**.**rst**(**rst**)**

**);**

**endmodule**

18、cpu\_try\_rule.txt

##INST\_1 0000 基本的逻辑指令和运算指令,更新标志位

ADD DR,SR "00000000[u4][u4]",DR,SR

SUB DR,SR "00000001[u4][u4]",DR,SR

DEC DR "00000010[u4]XXXX",DR

INC DR "00000011[u4]XXXX",DR

CMP DR,SR "00000110[u4][u4]",DR,SR

AND DR,SR "00000111[u4][u4]",DR,SR

OR DR,SR "00001000[u4][u4]",DR,SR

NOT DR "00001001[u4]XXXX",DR

XOR DR "00001010[u4]XXXX",DR

TEST DR,SR "00001011[u4][u4]",DR,SR

SHL DR "00001100[u4]XXXX",DR

SHR DR "00001101[u4]XXXX",DR

SAR DR "00001110[u4]XXXX",DR

#INST\_2 0010 立即数指令

LOADH IMM "00100000[u8]",IMM

LOADL IMM "00100001[u8]",IMM

#INST\_3 0100 堆栈指针

LOAD DR,SR "01000001[u4][u4]",DR,SR

STORE DR,SR "01000010[u4][u4]",DR,SR

#INST\_4 1100 NOP

NOP . "11000000XXXXXXXX"

#INST\_5 0011 HI LO指令 MOVE

MOVE DR,SR "00110000[u4][u4]",DR,SR

MOVZ DR,SR "00110001[u4][u4]",DR,SR

MOVN DR,SR "00110010[u4][u4]",DR,SR

MFHI DR "00110011[u4]XXXX",DR

MTHI SR "00110100XXXX[u4]",SR

MFLO DR "00110101[u4]XXXX",DR

MTLO SR "00110110XXXX[u4]",SR

#INST\_6 0101 乘除

MUL DR,SR "01010000[u4][u4]",DR,SR

MULU DR,SR "01010001[u4][u4]",DR,SR

DIV DR,SR "01010010[u4][u4]",DR,SR

DIVU DR,SR "01010011[u4][u4]",DR,SR

#INST\_7 0001 跳转指令

J IMM "00010000[u8]",IMM

JC IMM "00010001[u8]",IMM

JNC IMM "00010010[u8]",IMM

JZ IMM "00010011[u8]",IMM

JNZ IMM "00010100[u8]",IMM

R0 = 0

R1 = 1

R2 = 2

R3 = 3

R4 = 4

R5 = 5

R6 = 6

R7 = 7

R8 = 8

R9 = 9

R10= 10

R11= 11

R12= 12

R13= 13

R14= 14

R15= 15

19、inst.txt

MAIN: LOADH 0xff

LOAD R13,R0

MOVE R0,R15

ADD R1,R0

MOVE R4,R1

INC R4

STORE R14,R15

SUB R4,R1

OR R3,R1

AND R4,R3

NOT R3

TEST R3,R4

TEST R3,R3

CMP R4,R3

SHR R3

SAR R3

J 0x03

DEC R4

NOP

NOP

NOP

SHL R1

MOVZ R4,R1

MOVN R5,R1

MTLO R1

MFLO R7

MUL R0,R1

MFHI R11

MFLO R10

NOP

NOP

NOP

NOP

NOP

DIV R13,R11

MFHI R0

MFLO R1

NOP

NOP

NOP

NOP

NOP

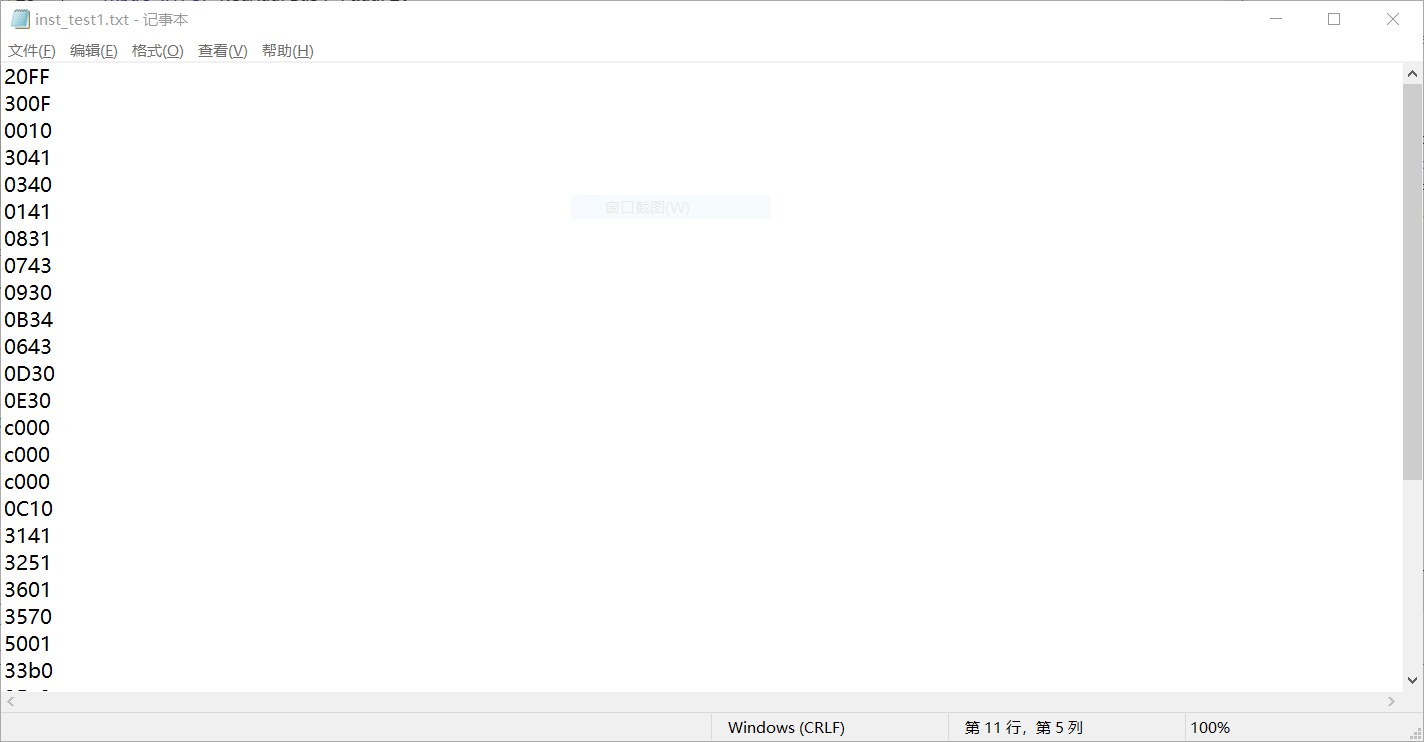
1. 遇到的问题及解决方法

1、调试困难的问题

注：由于仿真代码一个地址对应一个字节，故在仿真之前需将pc\_reg.v中的pc调整为每回合加4’h2才能一次读出16位。

面对复杂的控制信号以及时序数据传递，如果无法仿真的话，在出现问题时，就失去了寻找错误的最佳工具。在这次实验中，个人写了一个inst\_rom.v的文件，来模拟实验箱中的ram，inst\_rom.v有we写使能信号，16位地址总线，以及16位数据总线。inst\_rom中用寄存器组来模拟实际中的内存阵列。而寄存器组中的指令数据则使用

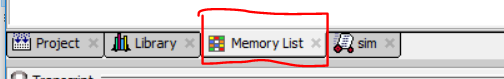
来进行初始化。其中inst\_test1.txt中即为指令的16进制数据。



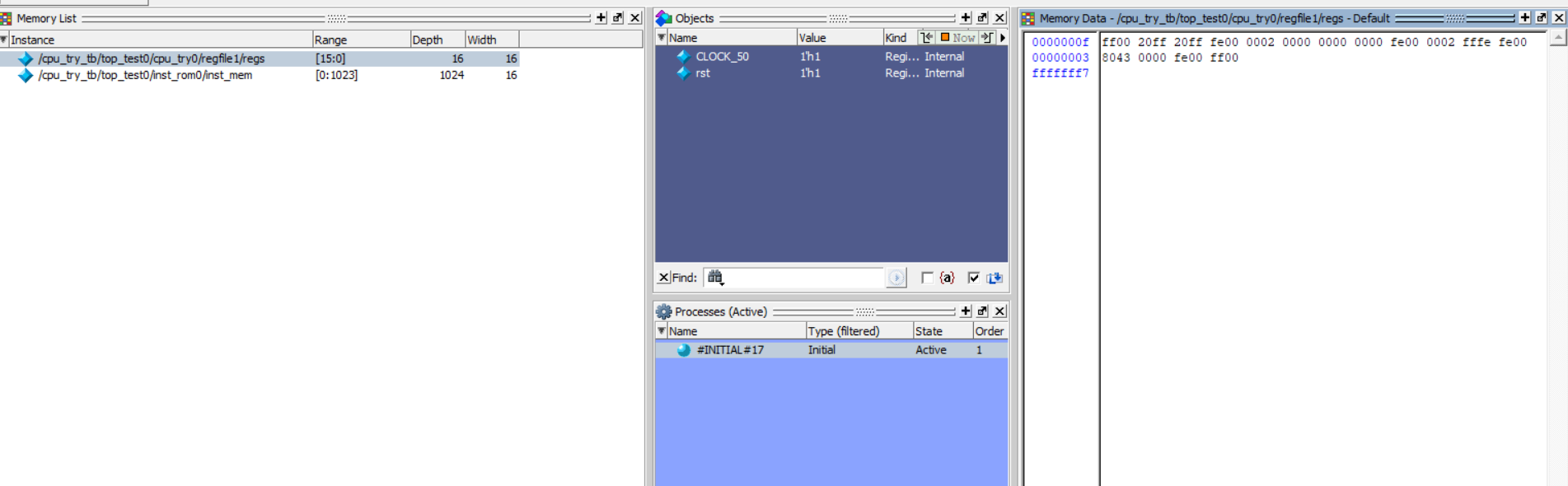
除此之外，通用寄存器组的数据为了仿真方便，也进行了相同的操作。但其是全为0000，即将通用寄存器组全部清零初始化，这在regfile.v中都可以看到。

除此之外，还需要一个更顶层的模块，将inst\_rom与cpu\_try模块相连接。这样最顶层模块top\_test接口只需要clk,rst即可。这样就可解决程序的调试困难问题。

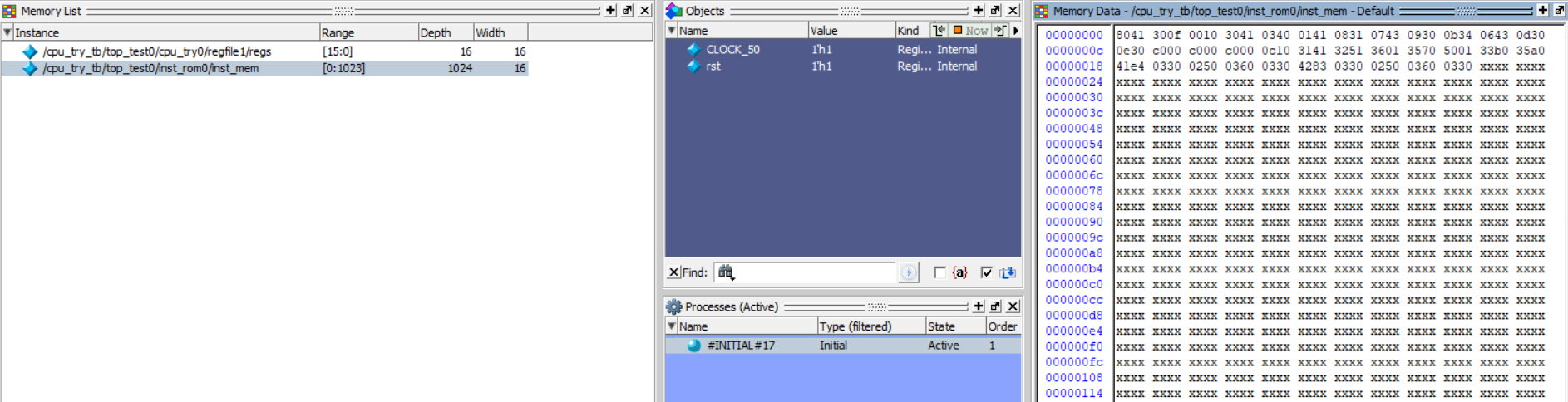
在用modelsim仿真时，除了可以观察波形之外，还可以清楚直观的看出通用寄存器组以及inst\_rom（最初文件命名有些问题，应该叫做inst\_ram，但后来由于太过麻烦，就没有改名）中的数据。



下图即为程序运行完成后通用寄存器组中的数据。

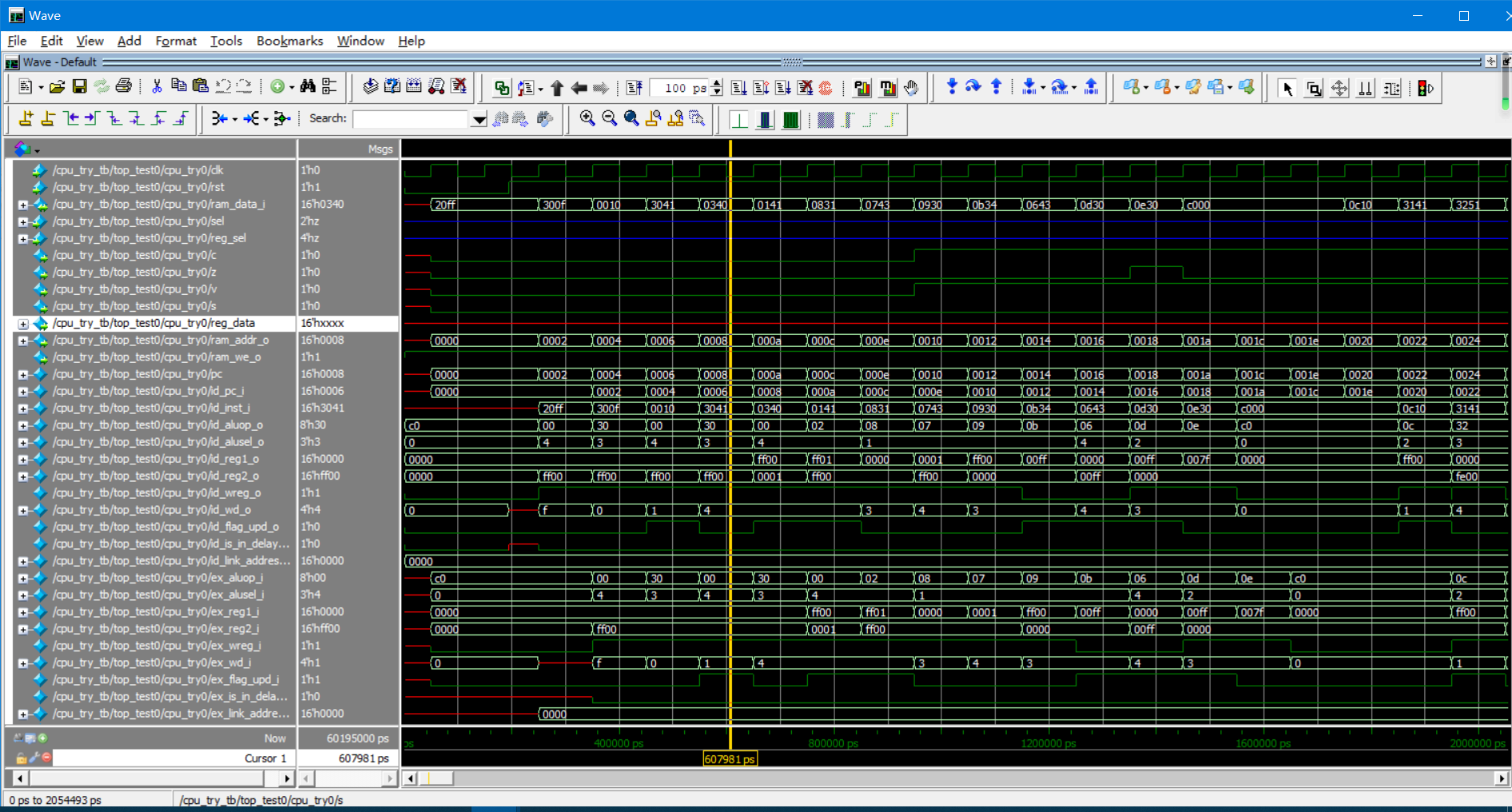


下图则为程序运行结束后模拟内存inst\_rom中的数据。



可以看到结果均符合预期。

下图为一次仿真示例，可以清晰看出流水线的运行，但是在实验箱上具体测试时，flag\_s标志位灯总是常亮，而仿真代码上s标志位没有异常，且检验代码多次后均为发现错误，故判断是硬件设备问题。



2、Inout端口没正确处理

在第一次上机实验时，出现了ram中的指令无法读入的问题。在课后与vhdl例程相比较，终于发现自己在inout端口没有正确配置，即应在读端口时，将inout端口赋值为高阻态，而在写数据时，将其赋值为要写的数据。

3、EX执行模块输出混乱

最初进行EX模块编写时，由于指令较少，就把所有指令的输出直接赋值给wdata\_o（要写入目的寄存器的值）。但当指令增多后，便需要各种判断来选择wdata\_o的赋值源。这极大的增加了电路综合后的复杂程度，显然不是合适的做法。

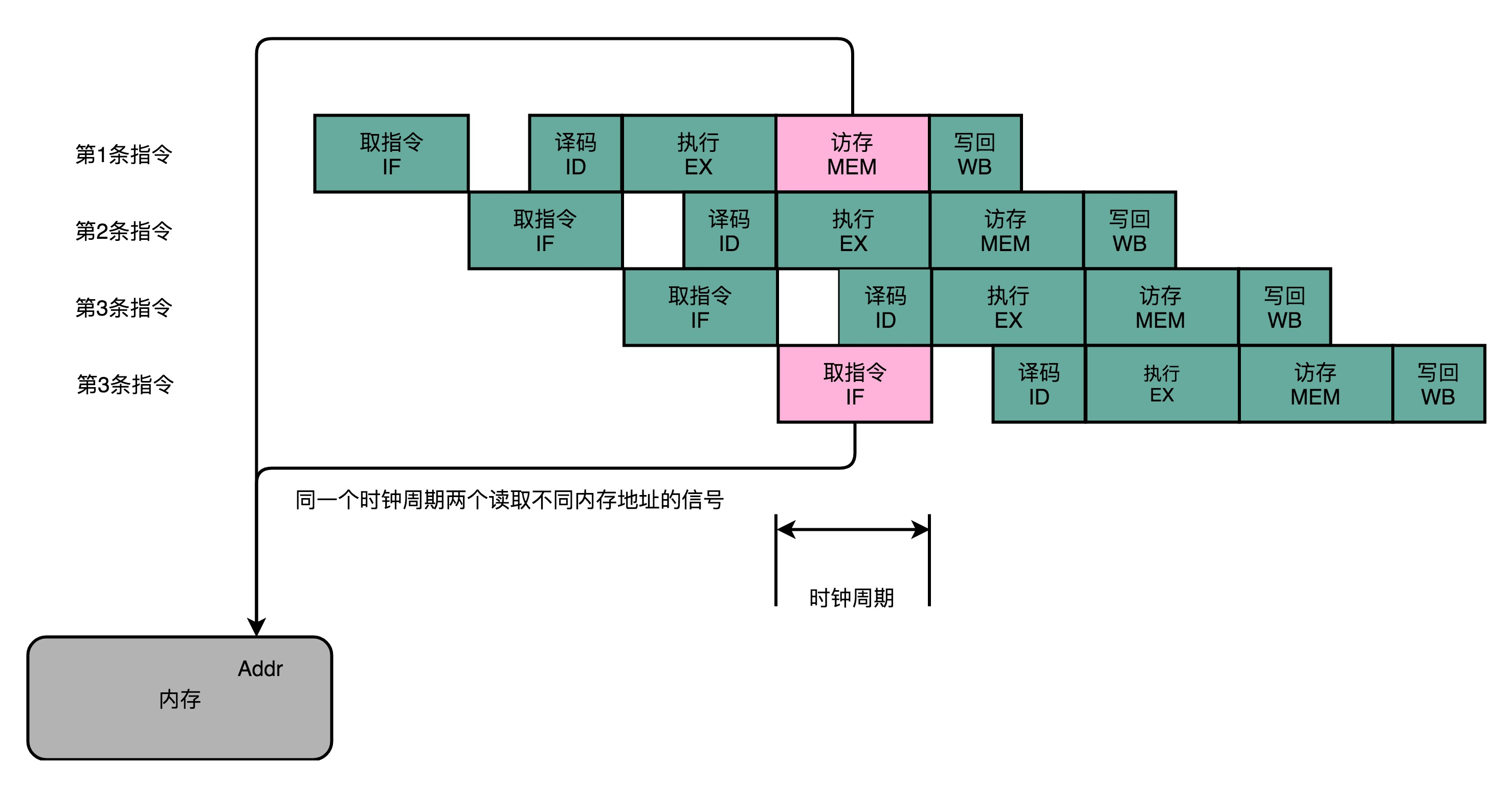
通过将指令的运算类型进行分类，而同一类型的指令，在执行的过程只会有一个结果，例如逻辑运算XOR,OR,AND等指令共用一个logicout来缓存结果，而乘法用mulres（注：mulresult）来缓存结果，以及除法div\_result等。

最终只需要选择指令的类型，即可进行数据的赋值，简化了代码的逻辑复杂度。

4、对imm的控制是否输出

由于在译码阶段就需要将imm立即数作为EX级运算的源操作数选择输出。个人在程序中规定，immenable作为立即数的使能信号，而一般立即数指令，无需源寄存器，故作出如下设定：reg2\_read\_o(寄存器第二个读端口的读使能信号)为0且immenable有效时，将imm作为运算级的源操作数2。

5、Load，Store 指令

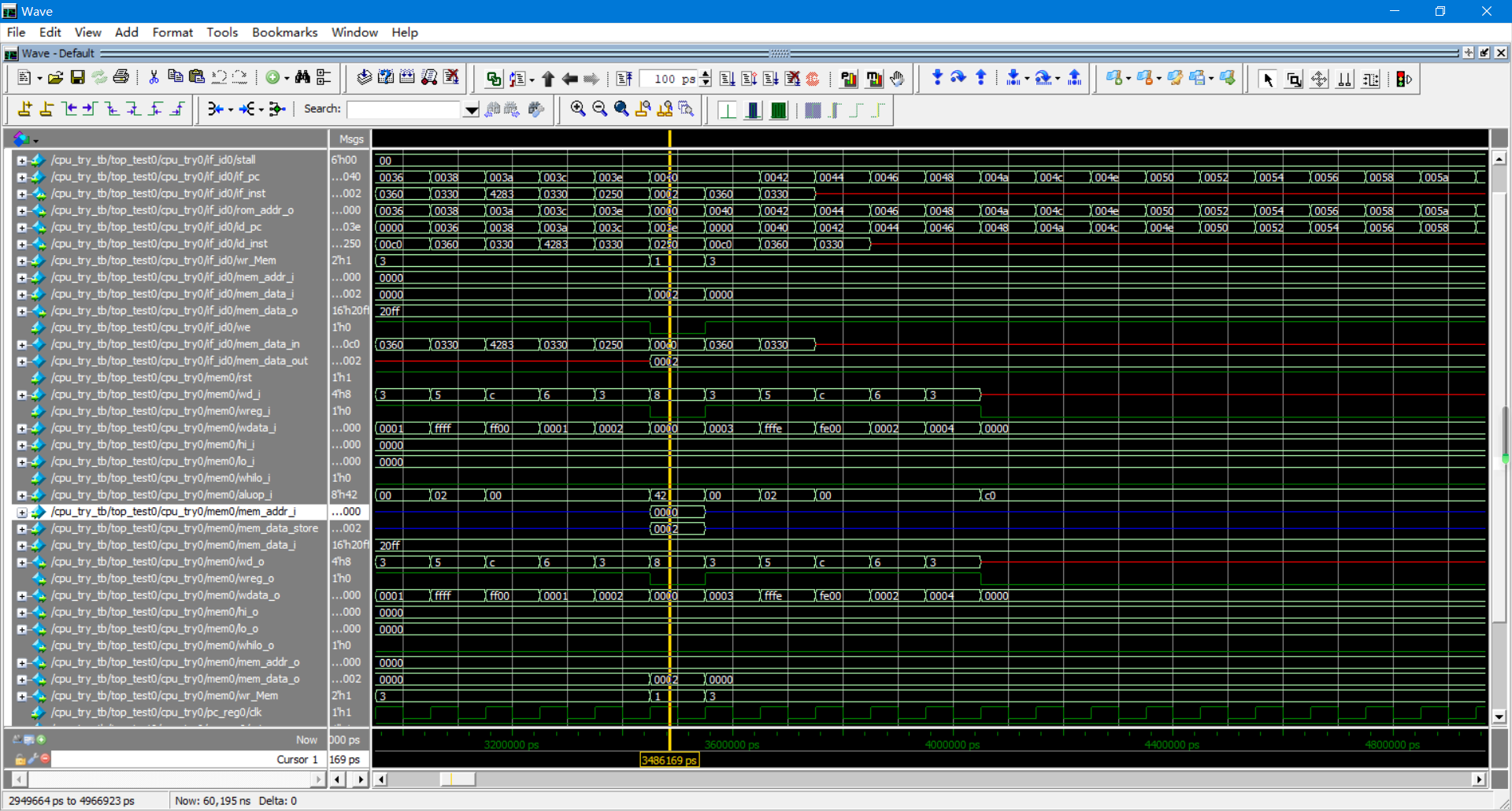


LOAD，STORE指令由于需要占用地址总线和数据总线，故须将pc阶段和IF/ID阶段暂停。而访存发生在两次时钟上升沿之间，故只需保持Pc\_reg的pc在访存下一个周期不变，同时在访存的下一个周期中IF/ID阶段输出的指令为NOP指令即可保证不影响访存前后取指的正常进行以及信号传递。

在Mem模块（访存模块）执行访存指令时，产生一个wr\_Mem信号（无需时钟上升沿，但仍为时序，故在下一个时钟到来aluop\_i(注：通过判断aluop\_i来判断是否为访存信号)改变之前，wr\_Mem不会改变，从而保护信号的稳定），并送至pc\_reg模块（在下一个时钟周期继续保持当前pc值，因为当前时钟周期，地址总线和数据总线被访存占用，故需下一周期继续取指。除此之外，并根据wr\_Mem指令对we信号更新并传到if/id模块）。

注：we在两模块中均为wire型。

当wr\_Mem信号传到IF/ID模块时，此时we信号也已更新完毕，且正处于下一个时钟上升沿之前，故进行数据的读取（组合逻辑），并将load出的数据（wire类型）传到mem模块。而mem模块中always(\*)则根据数据的改变进行wdata\_o的赋值。



此图即为store正常进行（黄色线段所在时钟周期）时wr\_Mem=2’b01时的仿真波形。可以看到在此阶段将mem\_data\_store=16’h0002送到了if\_inst数据总线。而在此时钟周期，if\_pc（pc值）保持不变，而在下一个周期继续执行。而在译码阶段向下一级传递NOP空指令。

6、处理Hazard

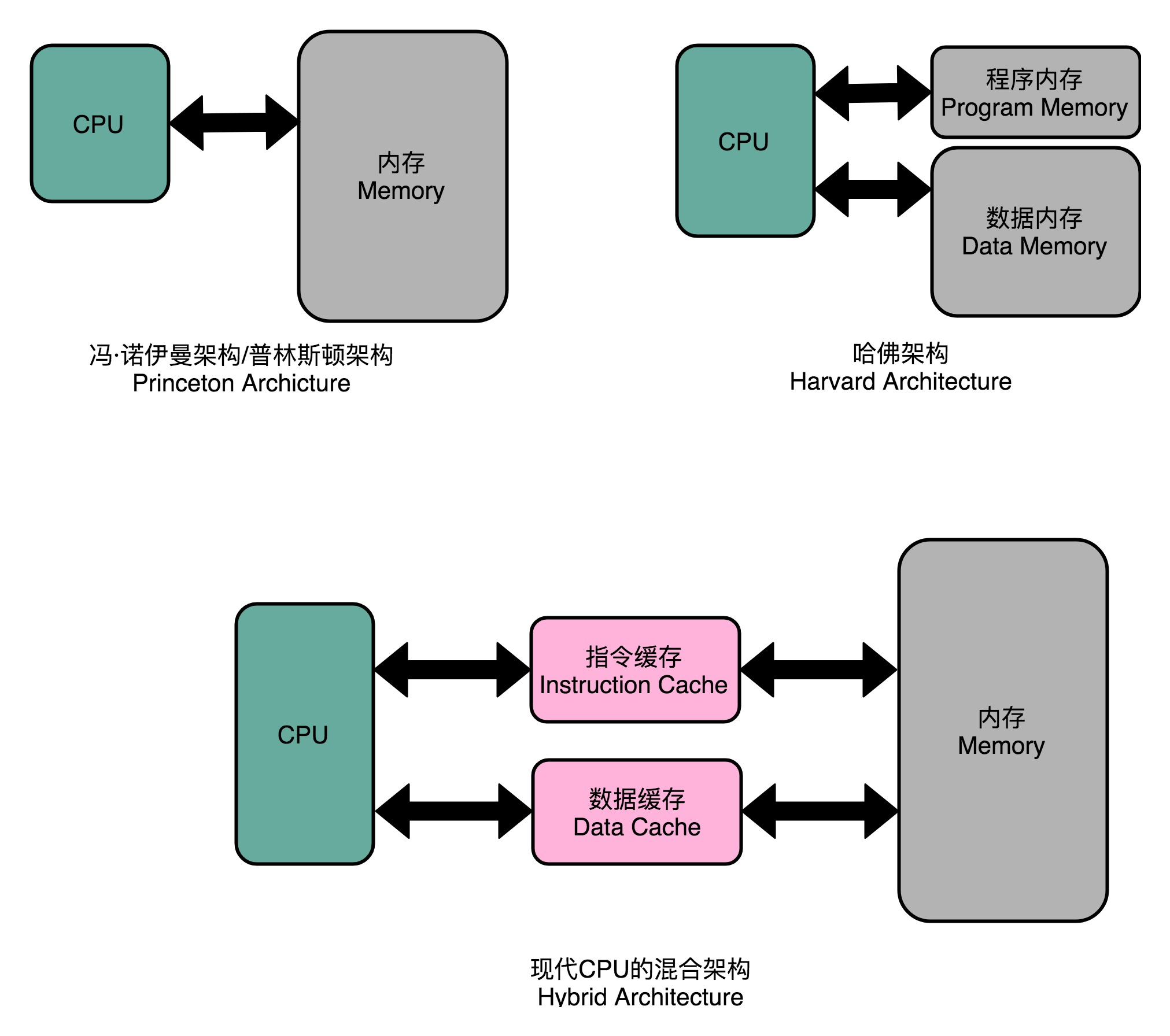
1、流水线架构提供了高吞吐量，但并不能简单地处理数据相关的问题。数据相关则会产生数据冒险的问题。

2、冒险的种类：

a.结构冒险（Structure Hazard）

结构冒险，本质上是一个硬件层面的资源竞争问题，也就是一个硬件电路层面的问题。CPU在同一个时钟周期，同时在运行两条计算机指令的不同阶段。但是这两个不同的阶段，可能会用到同样的硬件电路。最典型的例子就是内存的数据访问，即LOAD，STORE指令都要占用地址总线和数据总线。

对于资源冲突问题，其解决方案就是增加资源。对于访存和取指的冲突，一个直观的解决方案就是将内存分成两部分，分别有各自的地址译码器，分别存储指令和数据。而把内存分成两部分的解决方案，在计算机体系结构中称为哈佛架构(Harvard Architecture)。而不进行划分的即为冯诺依曼架构。



查阅资料得知，如今的CPU仍大多采用冯·诺伊曼架构，因为将内存划分成两部分后，就无法根据实际应用进行内存的动态分配， 使cpu的架构丧失了灵活性。

本课程设计仍采用冯·诺伊曼架构，通过插入空指令，将取指延后至访存指令（load，store）后一个时钟进行来处理结构冒险。

b.数据冒险（Data Hazard）

结构冒险是一个硬件层面的问题，我们可以靠增加硬件资源的方式来解决。然而还有很多冒险问题，是程序逻辑层面的事儿。其中，最常见的就是数据冒险。

数据冒险，其实就是同时在执行的多个指令之间，有数据依赖的情况。这些数据依赖，我们可以分成三大类，分别是 先写后读（Read After Write，RAW）、 先读后写（Write After Read，WAR）和 写后再写（Write After Write，WAW）。

即一条指令需要使用之前指令的结果，但是结果还没有写回。

RAW: 如指令 LOADH 0xff;

INC R15; //INC译码需要读R15数据

由于对INC译码时，R15的数据高8位0xff还未写入寄存器，故此时读出R15的数据不是最新值，进行后来运算就会出现错误。

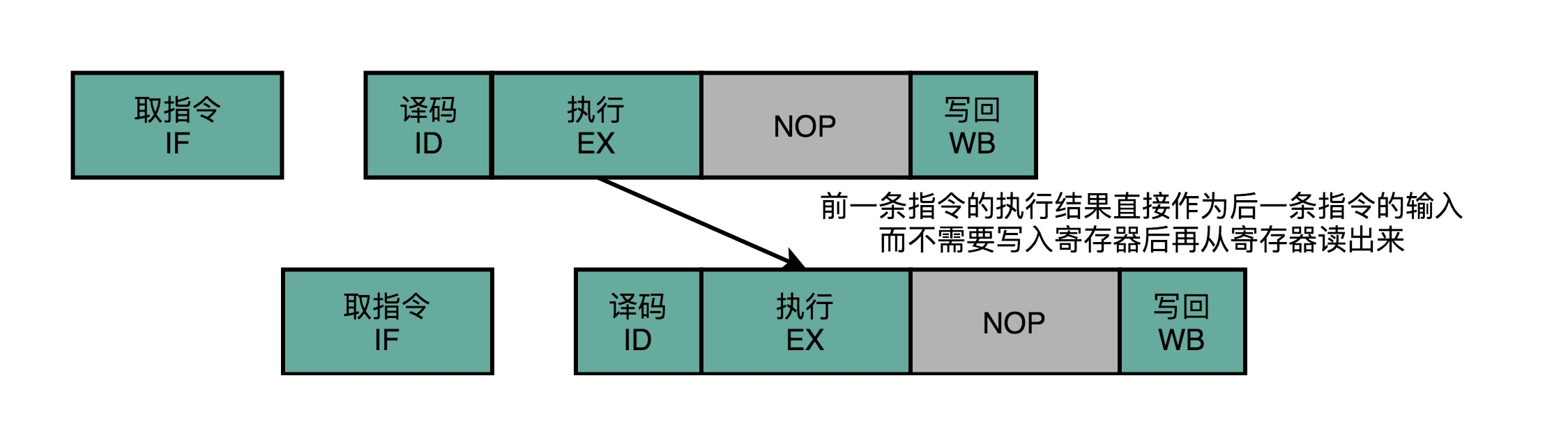
对于本课程设计，只有在流水线回写阶段才会写寄存器，所以不存在WAW相关；又因为只能在流水线译码阶段读寄存器、回写阶段才能写寄存器，所以不存在WAR相关，故只需考虑RAW相关即可。

解决方法：

1、插入NOP指令（软件编译层）。

2、流水线暂停（增加bubble-硬件层添加nop）。

3、数据前推 Data Forwarding（本课程设计采用方案）。



本课程设计中将EX执行级和MEM访存级以及MEM/WB回写级要写入目的寄存器的数据进行数据前推到ID译码级来选择最新的源操作数，默认EX优先级最高（EX级要写入目的寄存器的值为最新值），向后优先级递减进行逐级判断ID译码级要取的寄存器源操作数是否在前几条指令中执行了写操作。

除此之外，还有HI\_LO寄存器的数据前推，与通用寄存器组的前推同理，主要是在MFHI（move\_from\_hi）、MFLO（move\_from\_lo）指令执行时，可能会出写写后读数据相关（Read After Write, RAW）。

c.控制冒险（Control Hazard）

如果现在要执行哪条指令，是由之前指令的运行结果决定，而现在那条之前指令的结果还没产生，就导致了控制冒险。在本课程设计中主要是JC,JNC等跳转指令。问题便是尚未确定是否发生分支，如何进行下一次取指。

本课程设计采用延迟转移技术，如果在EX级（执行级）进行判断，如果进行分支跳转，则前两级（IF，ID级）的取指都会浪费，故将分支跳转指令判断提前到ID级（译码级），这样前方就只有一条指令存在着可能浪费的可能。这样，再进行规定，将JC, JZ, JNC, JNZ等指令的后一条指令称为延迟槽指令，处于延迟槽的指令无论是否跳转都必须执行，这样就可以通过软件层面的编译来处理控制冒险，这样在跳转指令执行时就不会存在资源的浪费。

七、课程设计体会及建议

在这次实验中，用verilog进行了一个五级流水线的复现，期间遇到了很多问题，但通过查阅资料，不断仿真调试，也收获了很多。尤其是流水线的五级模块具体实现时，可以和上学期的计算机原理中学习的cortex-M4以及其中的三级流水线有机结合起来。了解了当时不曾注意或者说不曾知晓的细微差别，当时自己在学习计算机原理时，还看了cortex-M4 manual Guide手册，了解了桶形移位器等的知识。这次在实验中，可惜自己时间精力有限，模块实现的都太过粗糙，控制信号还仍有较大冗余，或者说十分的丑陋，但自己仍然了解了很多计算机原理方面的知识，如一些除法的算法，普通的试商法，以及goldschmidt算法。除此之外，在查阅资料中，还了解了分支跳转有关的动态分支预测，以及多发射和超标量的知识，即实现指令的并行，但由于时间，也只是浅显的了解了一下。

通过这次实验，让自己对计算机原理的一些问题如流水线效率，流水线暂停机制的来源以及流水线每一个机制对应的问题解决方案有了更深的了解，但也让我更感受到自己仍有很多知识需要学习。

在验收时，老师也给了一些我曾经不曾想到的建议，如对时钟进行倍频，从而在一个主时钟中实现除法的运算，从而不需流水线的暂停。但这是与我最初的想法是不同的，我最初的想法，就是clk应为硬件层面所能容许的最大时钟频率，从而追求cpu的吞吐量最大化。而老师的建议则对应于主时钟频率较低且固定场景的解决方案。除此之外，老师的方法则可以通过使用alter的PLL IP核来例化PLL锁相环，才能写出倍频，但其仍存在频率稳定度的问题。通过集成了VCO的PLL（锁相环）输出的频率的稳定性与输入信号有关，优秀的PLL可以做到以内，一般的（大多数的）则在,这个偏差主要是由于跟踪输入频率引起。当PLL输出相位超前时，下一个周期会往后拉，如果相位滞后，则下一个周期往前拉，即jitter（时钟抖动）。但由于不知道实验箱的具体硬件参数，个人感觉使用这种方法的风险还是很大的。

自己在课程中工程量虽然很大，从架构上的模仿学习到细节处的信号传递以及指令的时钟周期控制，都需要反复的仿真验证，也写了一些冗余的代码，但总体来说还是收获了很多知识，对计算机的底层原理有了更深刻的理解，而且感觉自己对verilog代码的编写，以及modelsim仿真运用更加熟练。

课程建议：

1、课程只能在每次上课时才能进行测试，而且每个星期的两次课都在相邻的两天，这不利于代码的上机调试，而且在一天的间隔中，并不能写出太多有效的代码。故可以考虑每周相邻的两次课间隔3天或4天，来给学生提供足够的缓冲时间，进行代码编写。

2、实验箱的一些写信号以及双向端口（数据总线）对于初次接触的人有点难以理解。如inout端口在读数据时，需要将其赋值为全高阻，才能正确将数据读入。这可以在课程开始时对学生进行提醒。

3、在实验进行过程中，可能会对开题报告的一些不足之处进行改进，或者重新进行结构设计，故希望能将开题报告的比重与最终的设计报告相平衡一下权重，否则如果开题报告很可能会影响最终的设计方向，我在最初上机因为inout端口设置问题出错后，就因为开题报告的影响，也曾打算像别人一样在vhdl上直接修改。

八、分工

由于同组另一同学的个人原因，本工程由本人独立完成。