-Lab7 MIPS流水线CPU

1. 实现目标

* **设计一个MIPS五级流水线CPU**

1. 所实现的CPU功能

* **包含MIPS指令集36条指令**
* **实现五级流水，解决数据冒险、控制冒险**
* **对整个运算仿真**
* **实现下载，显示内存**

三．五级流水线设计

1. IF模块

* 功能：根据PC的值，从指令内存中读取一条指令，并设置下一

周期PC的值

* 构成：指令指针寄存器(PC)、指令存储器子模块(Instruction

ROM)、指令指针选择器(MUX) 、32位加法器

* 指令指针选择器：

|  |  |
| --- | --- |
| 地址 | PC来源 |
| {JR,J,Z}=100 | JR地址 |
| {JR,J,Z}=010 | J地址 |
| {JR,J,Z}=001 | Branch地址 |
| {JR,J,Z}=000 | PC+4 |

* PC寄存器

当发生数据冒险时，需保持PC寄存器不变，PC使能信号为

PC\_IFWrite

2. ID模块

* 功能：解析指令的操作码，产生各种控制信号。流水线冒险检测

也在ID级进行，冒险检测电路需要上一条指令的

MemRead, 在检测到冒险条件成立时，冒险检测电路

产生stall信号，清空ID/EX寄存器，插入一个流水线气

泡

* 构成：指令译码(Decode)、寄存器堆、冒险检测、分支检测、

加法器

* 指令译码(Decode)

输入指令(共36条):

|  |  |  |
| --- | --- | --- |
| R型 | type1 | ADD, ADDU, SUB, SUBU, AND, OR, NOR, XOR, SLT,  SLTU,SLLV,SRLV, SRAV |
| type2 | SLL, SRL, SRA |
| JR | |
| J型 | J | |
| I型 | type1 | ADDI, ADDIU, ANDI, ORI, XORI, SLTI, SLTIU |
| Branch | BEQ, BNE, BGEZ, BGTZ, BLEZ, BLTZ |
| LW,LH,LB | |
| SW | |

输出信号：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | RegWrite | 是否写寄存器 | LW, R-type1, R-type2, I-type1 |
| 2 | RegDst | 目标寄存器是rt还是rd | rd(RegDst=1): R-type1, R-type2  rt(RegDst=0): LW, I-type1 |
| 3 | MemWrite | 是否写数据存储器 | SW |
| 4 | MemRead | 是否读数据存储器 | LW |
| 5 | MemtoReg | 写寄存器的数据来自Mem还是ALU | ALU(MmetoReg=0): R-type1, R-type2, I-type1  Mem(MemtoReg=1): LW |
| 6 | ALUSrcA | ALU第一操作数为rs还是sa | rs(ALUSrcA=0):LW, R-type1, I-type1  sa(ALUSrcA=1): R-type2 |
| 7 | ALUSrcB | ALU第二操作数为rt还是Imm | rt(ALUSrcB=0): R-type1, R-type2  Imm(ALUSrcB=1):LW,SW,I-type1 |
| 8 | PCSrc | 写入PC寄存器的来源是PC+4,Branch地址,J地址还是JR地址 | PC+4(PCSrc=000): 其余  Branch(PCSrc=001): Branch  J(PCSrc=010): J  JR(PCSrc=100): JR |
| 9 | LwByte | 写入 |  |
| 10 | ALUCode | 决定ALU功能，由指令中的op段,rt段,funct段共同决定 | |

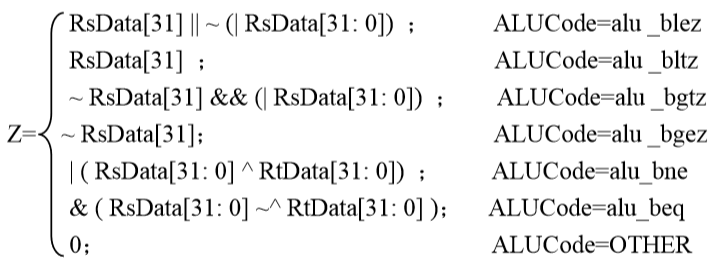
ALUCode译码表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| op | funct | rt | 运算 | ALUCode |
| BEQ\_op | XXX | XXX | Z=(A==B) | 01010 /10 |
| BNE\_op | XXX | XXX | Z=~(A==B) | 01011 /11 |
| BGEZ\_op | XXX | 0 | Z=(A>=0) | 01100 /12 |
| BGTZ\_op | XXX | 0 | Z=(A>0) | 01101 /13 |
| BLEZ\_op | XXX | 0 | Z=(A<=0) | 01110 /14 |
| BLTZ\_op | XXX | 0 | Z=(A<0) | 01111 /15 |
| R-type\_op | ADD\_funct | XXX | 加 | 00000 /0 |
| ADDU\_funct | XXX | 10101 /21 |
| AND\_funct | XXX | 与 | 00001 /1 |
| XOR\_funct | XXX | 异或 | 00010 /2 |
| OR\_funct | XXX | 或 | 00011 /3 |
| NOR\_funct | XXX | 或非 | 00100 /4 |
| SUB\_funct | XXX | 减 | 00101 /5 |
| SUBU\_funct | XXX | 10110 /22 |
| SLT\_funct | XXX | A<B?1:0 | 10011 /19 |
| SLTU\_funct | XXX | 10100 /20 |
| SLL\_funct | XXX | 左移 | 10000 /16 |
| SLLV\_funct | XXX | 10000 /16 |
| SRL\_funct | XXX | 逻辑右移 | 10001 /17 |
| SRLV\_funct | XXX | 10001 /17 |
| SRA\_funct | XXX | 算术右移 | 10010 /18 |
| SRAV\_funct | XXX | 10010 /18 |
| ADDI\_op | XXX | XXX | 加 | 00000 /0 |
| ADDIU\_op | XXX | XXX | 00000 /0 |
| ANDI\_op | XXX | XXX | 与 | 00110 /6 |
| XORI\_op | XXX | XXX | 异或 | 00111 /7 |
| ORI\_op | XXX | XXX | 或 | 01000 /8 |
| SLTI\_op | XXX | XXX | A<B?1:0 | 10011 /19 |
| SLTIU\_op | XXX | XXX | 10100 /20 |
| SW\_op | XXX | XXX | 加 | 00000 /0 |
| LW\_op | XXX | XXX | 00000 /0 |

* 分支检测(Branch Test)

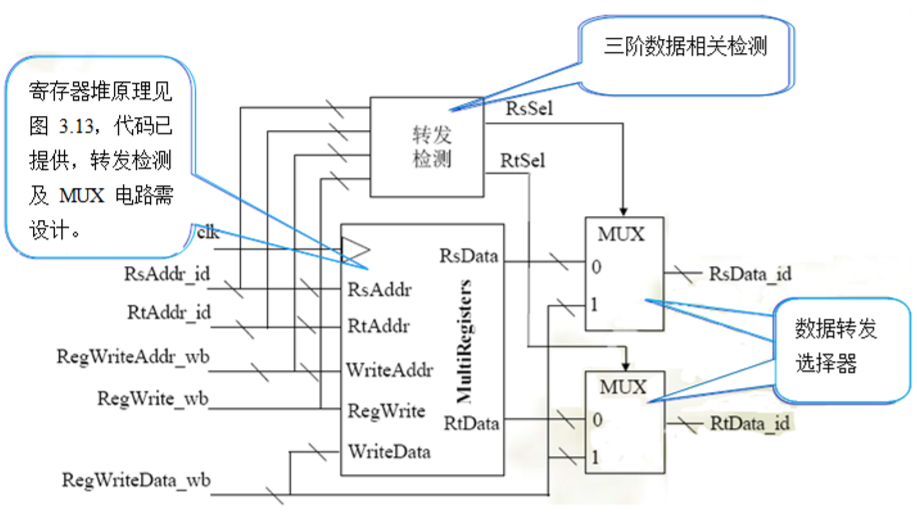
Zero检测Branch条件是否成立，其中BEQ、BNE 两个操作数为 RsData 与 RtData，而 BGEZ、BGTZ、BLEZ 和 BLTZ 指令则为 RsData 与常数 0 比较。

输出信号Zero表达式如下



* 寄存器堆(Registers)

在流水线 CPU 设计中，寄存器堆设计还应解决三阶数据相关的数据转发问题。当满足三阶数据相关条件时，寄存器具有 Read after Write 的特性。为实现该功能，在寄存器堆的基础上加一转发电路。

RsSel=RegWrite\_wb&&(~(RegWriteAddr\_wb==0))&&(RegWriteAddr\_wb==RsAddr\_id) RtSel=RegWrite\_wb&&(~(RegWriteAddr\_wb==0))&&(RegWriteAddr\_wb==RtAddr\_id) 

* 冒险检测(Hazard Detector)

冒险成立的条件：

(1)上一条指令为LW，即MemRead\_ex=1;

(2)在EX级的LW指令与在ID级的指令读写的是同一个寄存器，即

RegWriteAddr\_ex=Rs\_Addr\_id/RtAddr\_id;

解决冒险的方法：

(1) 插入一个流水线气泡 Stall 清空 ID/EX 寄存器并且阻塞流水线 ID

级、IF 级流水线

Stall=((RegWriteAddr\_ex==RsAddr\_id) ||

(RegWriteAddr\_ex==RtAddr\_id)) &&MemRead\_ex

(2) 保持 PC 寄存器和 IF/ID 流水线寄存器不变

PC\_IFWrite=~Stall;

* 其他部分

1. Branch指令分支地址计算

BranchAddr=NextPC\_id+(sign-extend(Imm\_id)<<2)

1. JR指令跳转地址计算

JRAddr=RsData\_id

1. J指令跳转地址计算

JAddr={NextPC\_id[31:28],IR\_id[25:0],2’b00}

1. 符号扩展

有符号数补符号位，无符号数补0

3. EX模块

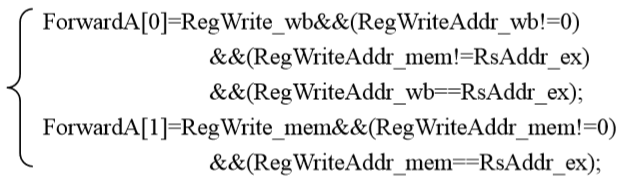
* 功能: 执行ALU运算和标志寄存器设置
* 构成：ALU 、转发电路 Forwarding 、数据选择器
* ALU设计
* 转发电路(Forwarding)

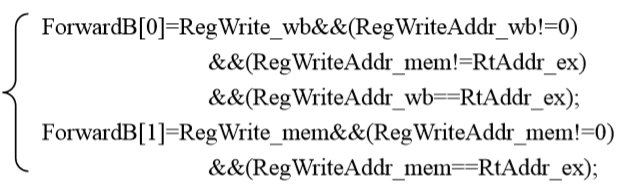
操作数 A 和 B 由数据选择器决定，数据选择器的地址信号即为

ForwardA 和 ForwardB。 其含义如下表:

|  |  |  |
| --- | --- | --- |
| 地址 | 操作数来源 | 说明 |
| ForwardA=00 | RsData\_ex | 操作数 A 来自寄存器堆 |
| ForwardA=01 | RegWriteData\_wb | 操作数 A 来自二阶数据相关的转发数据 |
| ForwardA=10 | ALUresult\_mem | 操作数 A 来自一阶数据相关的转发数据 |
| ForwardA=00 | RtData\_ex | 操作数 B 来自寄存器堆 |
| ForwardA=01 | RegWriteData\_wb | 操作数 B 来自二阶数据相关的转发数据 |
| ForwardA=10 | ALUresult\_mem | 操作数 B 来自一阶数据相关的转发数据 |

由一、二阶数据相关判断条件：





4. MEM模块

* 功能: MEM阶段要根据指令功能和上一阶段的运算结果(内存操作的时候作为内存地址), 决定是否要访问内存以及如何访问。只对需要内存操作的指令有效。
* 构成：数据存储器

5.WB模块

* 功能: 根据指令的功能以及上一阶段的结果，决定是否要修改寄存器的值以及如何修改，只对需要修改寄存器值的指令有效。
* 构成：数据选择器

四．寄存器设计

流水线寄存器负责将流水线的各部分分开，共有 IF/ID、ID/EX、

EX/MEM、MEM/WB 四组。

(1)IF/ID：当发生数据冒险时，需保持IF/ID流水线寄存器不变，

故IF/ID流水线寄存器具有使能信号PC\_IFWrite输入；

当流水线发生分支冒险时，需清空IF/ID流水线寄存器，

清零信号为IF\_flush。

(2)ID/EX：当流水线发生数据冒险时，需清空ID/EX流水线寄存

器而插入一个气泡。

(3)EX/MEM：普通D型寄存器

(4)MEM/WB：普通D型寄存器

1. 冒险的处理
2. 数据相关—转发机制的实现

* 一阶数据相关(EX冒险)

(1)定义：第i条指令的源操作数与第i-1条指令的目标寄存器相重

(2)例：

|  |
| --- |
| **Sub** $2**,** $1**,** $3  **And** $12**,** $2**,** $5 |

(3)分析：sub的指令结果在第三周期末产生，and在第四周期向sub的

结果发出请求

(4)解决方法：sub 指令结果产生之后直接将其转发给and指令，转发的数

据为ALUResult\_mem

(5)转发条件：①MEM级指令为写操作,即RegWrite\_mem=1;

②MEM级指令写回目标寄存器与EX级指令源寄存器是同一

寄存器,即RegWriteAddr\_mem=RsAddr\_ex/RtAddr\_ex

* 二阶数据相关(MEM冒险)

(1)定义：第i条指令的源操作数与第i-2条指令的目标寄存器相重

(2)例：

|  |
| --- |
| **sub** $2**,** $1**,** $3  **and** $12**,** $2**,** $5  **or** $13**,** $6**,** $2 |

(3)分析：sub指令在第5周期写回寄存器，or指令在第五周期对sub指令

的结果提出请求

(4)解决方法：通过MEM/WB流水线寄存器，将sub指令结果转发给or,

转发数据为RegWriteData\_wb

(5)转发条件：①WB级指令是写操作，即RegWrite\_wb=1;

②WB级指令写回的目标寄存器与在EX级指令的源寄存器

是同一寄存器，即RegWriteAddr\_wb=RsAddr\_ex或

RegWriteAddr\_wb=RtAddr\_ex

③EX冒险不成立，即RegWriteAddr\_mem≠RsAddr\_ex

或RegWriteAddr\_mem≠RtAddr\_ex

* 三阶数据相关

(1)定义：同一周期内同时读写同一个寄存器

(2)例：

|  |
| --- |
| **sub** $2**,** $1**,** $3  **and** $12**,** $2**,** $5  **or** $13**,** $6**,** $2  **add** $14**,** $2**,** $2 |

(3)分析：sub指令和add指令在第5周期同时读写$2

(4)解决方法：通过MEM/WB流水线寄存器，将sub指令结果转发给add,

转发数据为RegWriteData\_wb

(5)转发条件：①WB级指令是写操作，即RegWrite\_wb=1;

②WB级指令写回的目标寄存器与在ID级指令的源寄存器

是同一寄存器，即RegWriteAddr\_wb=RsAddr\_id或

RegWriteAddr\_wb=RtAddr\_id

1. 数据冒险—冒险检测电路的实现

* 定义：当一条指令试图读取一个寄存器，而它前一条指令是 lw 指

令，并且该 lw 指令写入的是同一个寄存器时，定向转发的

方法就无法解决问题

* 解决方法【冒险检测+流水线阻塞】

1. 冒险成立条件：①上一条指令是lw指令，即MemRead\_ex=1;

②在EX级的lw指令与在ID级的指令读写的是同一个

寄存器，即RegWriteAddr\_ex=RsAddr\_id或

RegWriteAddr\_ex=RtAddr\_id

1. 冒险的检测：在ID级插入冒险检测电路(Hazard Detector)，当冒险条件成立时，在

Lw指令和下一条指令间插入阻塞，使后一条指令延迟一个时钟周期执行，这样就将该冒险转化为二阶数据相关，可用转发解决。

(3)流水线阻塞的实现：

①在 ID 级检测到冒险条件时， HazardDetector 输出 两个信号：

Stall 与 PC\_IFWrite

②Stall 信号将 ID/EX 流水线寄存器中的 EX、MEM 和 WB 级控制

信号全部清零。这些信号传递到流水线后 面的各级，由于控制信号均

为零，所以不会对任何寄存 器和存储器进行写操作，高电平有效。

③PC\_IFWrite 信号禁止 PC 寄存器和 IF/ID 流水线寄 存器接收新数

据，低电平有效

1. 控制冒险

* 定义：流水线每个时钟周期都得取指令才能维持运行，但分支指令必须等到 MEM 级才能确定是否执行分支。这种为了确定预取正确的指令而导致的延迟叫做控制冒险或分支冒险。
* 解决方法【提前分支指令】

1. 计算分支的目的地址:

由于已经有了 PC 值和 IF/ID 流水线寄存器中的指令值，所以可以很方便地将 EX 级的分支地址计算电路移到 ID 级。故针对所有指令都执行分支地址的计算过程，但只有在需要它的时候才会用到。

1. 判断分支指令的跳转条件:

将用于判断分支指令成立的 Zero 信号检测电 路（Z test ）从 ALU 中独立出来，并将它从 EX 级提前 至 ID 级。

1. 丢弃 IF 级 的指令:

加入一个控制信号 IF\_flush，做 为 IF/ID 流水线寄存器的清零信号。当分支冒险成立， 即 Z=1，则 IF\_flush=1，否则 IF\_flush=0，故 IF\_flush = Z。考虑到本系统还要实现的无条件跳转指令： J 和 JR， 在执行这两个指令时也必须要对 IF/ID 流水线寄存器进 行清空，因此， IF\_flush 的表达式应表示为：IF\_flush = Z || J || JR

1. 实验分析

流水线CPU的重难点主要在：

(1)MIPS指令的分类和处理; (2)数据冒险和控制冒险的处理，

具体体现在代码上就是Deocde模块的设计和Forwarding模块、HazardDetector模块的设计；(3)各个模块的连接；

1. 源代码(五级流水线组织部分)

|  |
| --- |
| `timescale 1ns **/** 1ps  ////////////////////////////////////////////////////////////////  // Company:  // Engineer:  //  // Create Date: 21:23:26 06/05/2018  // Design Name: ZeroTest  // Module Name: E:/DownLoad/COD\_lab7/CPU.v  // Project Name: COD\_lab7  // Target Device:  // Tool versions:  // Description:  //  //  // Dependencies:  //  // Revision:  // Revision 0.01 - File Created  // Additional Comments:  //  //////////////////////////////////////////////////////  **module** CPU**;**  **reg** clk**;**  **initial**  **begin**  clk**=**0**;**  **#**10**;**  **forever** **#**5 clk**=~**clk**;**  **end**  ////////////////////////////【IF】//////////////////////////////  **wire** **[**31**:**0**]**NextPC\_if**,**BranchAddress**,**JumpAddress**,**RsData\_id**;**  **reg** **[**31**:**0**]**PC\_in**;**  **wire** Z**,**J**,**JR**;**  **wire** RegWrite\_ex**;**  **wire** **[**4**:**0**]**RegWriteAddr\_ex**,**RsAddr\_id**;**  **wire** **[**31**:**0**]**ALUResult\_ex**;**  **wire** **[**31**:**0**]**JrAddr**;**  **wire** forward\_jr**;**  **assign** forward\_jr**=**RegWrite\_ex**&&(**RegWriteAddr\_ex**==**RsAddr\_id**);**  **assign** JrAddr**=**forward\_jr **?** ALUResult\_ex **:** RsData\_id**;**    //MUX  **always@(\*)**  **begin**  **case({**JR**,**J**,**Z**})**  3'b000**:**PC\_in**<=**NextPC\_if**;**  3'b001**:**PC\_in**<=**BranchAddress**;**  3'b010**:**PC\_in**<=**JumpAddress**;**  3'b100**:**PC\_in**<=**JrAddr**;**  **default:**PC\_in**<=**32'b0**;**  **endcase**  **end**  //PC  **wire** **[**31**:**0**]**PC**;**  **assign** NextPC\_if**=**PC**+**4**;**  **wire** PC\_IFWrite**;**  Reg **#(.**width**(**32**))**PC\_reg\_if**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**PC\_IFWrite**),.**in**(**PC\_in**),.**out**(**PC**));**    //InstructionRom  **wire** **[**31**:**0**]**Instruction\_if**;**  InstructionROM InstrUnit**(.**addr**(**PC**),.**dout**(**Instruction\_if**));**    //FLUSH  **wire** IF\_flush**;**  **assign** IF\_flush**=**JR**||**J**||**Z**;**  //---------------------------【IF/ID】--------------------------------  **wire** **[**31**:**0**]**NextPC\_id**;**  **wire** **[**31**:**0**]**Instruction\_id**;**  Reg **#(.**width**(**32**))**PC\_if\_id**(.**clk**(**clk**),.**reset**(**IF\_flush**),.**enable**(**PC\_IFWrite**),.**in**(**NextPC\_if**),.**out**(**NextPC\_id**));**  Reg **#(.**width**(**32**))**Instr\_if\_id**(.**clk**(**clk**),.**reset**(**IF\_flush**),.**enable**(**PC\_IFWrite**),.**in**(**Instruction\_if**),.**out**(**Instruction\_id**));**  //--------------------------------------------------------------------------------------------------------------------------  ////////////////////////////////【ID】///////////////////////////////  //指令译码  **wire** MemtoReg\_id**,**RegWrite\_id**,**MemWrite\_id**,**MemRead\_id**,**ALUSrcA\_id**,**ALUSrcB\_id**,**RegDst\_id**,**Branch\_id**;**  **wire** **[**1**:**0**]**LwByte\_id**;**  **wire** **[**4**:**0**]**ALUCode\_id**;**  Decode decoder**(.**Instruction**(**Instruction\_id**),.**MemtoReg**(**MemtoReg\_id**),.**RegWrite**(**RegWrite\_id**),.**MemWrite**(**MemWrite\_id**),.**MemRead**(**MemRead\_id**),.**ALUCode**(**ALUCode\_id**),**  **.**ALUSrcA**(**ALUSrcA\_id**),.**ALUSrcB**(**ALUSrcB\_id**),.**RegDst**(**RegDst\_id**),.**J**(**J**),.**JR**(**JR**),.**Branch**(**Branch\_id**),.**LwByte**(**LwByte\_id**));**  **wire** **[**4**:**0**]**RtAddr\_id**,**RdAddr\_id**;**  **assign** RsAddr\_id**=**Instruction\_id**[**25**:**21**];**  **assign** RtAddr\_id**=**Instruction\_id**[**20**:**16**];**  **assign** RdAddr\_id**=**Instruction\_id**[**15**:**11**];**  //位扩展  **wire** **[**31**:**0**]**Imm\_id**,**Sa\_id**;**  **assign** Imm\_id**={{**16**{**Instruction\_id**[**15**]}},**Instruction\_id**[**15**:**0**]};**  **assign** Sa\_id **={**27'b0**,**Instruction\_id**[**10**:**6**]};**    //分支跳转地址计算  **assign** BranchAddress**=**NextPC\_id**+(**Imm\_id**<<**2**);**  **assign** JumpAddress**={**NextPC\_id**[**31**:**28**],**Instruction\_id**[**25**:**0**],**2'b00**};**  //分支检测  **wire** **[**31**:**0**]**RtData\_id**;**  **wire** forward\_zero\_Rs**,**forward\_zero\_Rt**;**  **assign** forward\_zero\_Rs**=**RegWrite\_ex**&&(**RegWriteAddr\_ex**==**RsAddr\_id**);**  **assign** forward\_zero\_Rt**=**RegWrite\_ex**&&(**RegWriteAddr\_ex**==**RsAddr\_id**);**  **wire** **[**31**:**0**]**RsData\_zero**,**RtData\_zero**;**  **assign** RsData\_zero**=**forward\_zero\_Rs **?** ALUResult\_ex **:** RsData\_id**;**  **assign** RtData\_zero**=**forward\_zero\_Rt **?** ALUResult\_ex **:** RtData\_id**;**  ZeroTest zero\_unit**(.**ALUCode**(**ALUCode\_id**),.**RsData**(**RsData\_zero**),.**RtData**(**RtData\_zero**),.**Z**(**Z**));**    //冒险检测  **wire** MemRead\_ex**;**  **wire** Stall**;**  HazardDetector hazard\_unit**(.**RegWriteAddr**(**RegWriteAddr\_ex**),.**MemRead**(**MemRead\_ex**),.**RsAddr**(**RsAddr\_id**),.**RtAddr**(**RtAddr\_id**),.**Stall**(**Stall**),.**PC\_IFWrite**(**PC\_IFWrite**));**  //寄存器堆  **wire** RegWrite\_wb**;**  **wire** **[**4**:**0**]**RegWriteAddr\_wb**;**  **wire** **[**31**:**0**]**RegWriteData\_wb**;**  **wire** **[**31**:**0**]**RsData\_temp**,**RtData\_temp**;**  Registers register\_unit**(.**clk**(**clk**),.**RsAddr**(**RsAddr\_id**),.**RtAddr**(**RtAddr\_id**),.**WriteData**(**RegWriteData\_wb**),.**WriteAddr**(**RegWriteAddr\_wb**),.**RegWrite**(**RegWrite\_wb**),**  **.**RsData**(**RsData\_temp**),.**RtData**(**RtData\_temp**));**  //三阶转发处理  **wire** RsSel**,**RtSel**;**  **assign** RsSel**=**RegWrite\_wb**&&(~(**RegWriteAddr\_wb**==**0**))&&(**RegWriteAddr\_wb**==**RsAddr\_id**);**  **assign** RtSel**=**RegWrite\_wb**&&(~(**RegWriteAddr\_wb**==**0**))&&(**RegWriteAddr\_wb**==**RtAddr\_id**);**  **assign** RsData\_id**=(**RsSel**==**1**)?** RegWriteData\_wb **:** RsData\_temp**;**  **assign** RtData\_id**=(**RtSel**==**1**)?** RegWriteData\_wb **:** RtData\_temp**;**    //----------------------------【ID/EX】----------------------------  **wire** MemtoReg\_ex**,**MemWrite\_ex**,**RegDst\_ex**,**ALUSrcA\_ex**,**ALUSrcB\_ex**;**  **wire** **[**1**:**0**]**LwByte\_ex**;**  **wire** **[**4**:**0**]**ALUCode\_ex**;**  **wire** **[**31**:**0**]**Sa\_ex**,**Imm\_ex**,**RsData\_ex**,**RtData\_ex**;**  **wire** **[**4**:**0**]**RdAddr\_ex**,**RtAddr\_ex**,**RsAddr\_ex**;**  Reg **#(.**width**(**2**))**LwByte\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**LwByte\_id**),.**out**(**LwByte\_ex**));**  Reg **#(.**width**(**2**))**WB\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**({**MemtoReg\_id**,**RegWrite\_id**}),.**out**({**MemtoReg\_ex**,**RegWrite\_ex**}));**  Reg **#(.**width**(**2**))**M\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**({**MemWrite\_id**,**MemRead\_id**}),.**out**({**MemWrite\_ex**,**MemRead\_ex**}));**  Reg **#(.**width**(**3**))**EX\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**({**RegDst\_id**,**ALUSrcA\_id**,**ALUSrcB\_id**}),.**out**({**RegDst\_ex**,**ALUSrcA\_ex**,**ALUSrcB\_ex**}));**  Reg **#(.**width**(**5**))**ALUCode\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**ALUCode\_id**),.**out**(**ALUCode\_ex**));**  Reg **#(.**width**(**32**))**Sa\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**Sa\_id**),.**out**(**Sa\_ex**));**  Reg **#(.**width**(**32**))**Imm\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**Imm\_id**),.**out**(**Imm\_ex**));**  Reg **#(.**width**(**5**))**RdAddr\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**RdAddr\_id**),.**out**(**RdAddr\_ex**));**  Reg **#(.**width**(**5**))**RsAddr\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**RsAddr\_id**),.**out**(**RsAddr\_ex**));**  Reg **#(.**width**(**5**))**RtAddr\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**RtAddr\_id**),.**out**(**RtAddr\_ex**));**  Reg **#(.**width**(**32**))**RsData\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**RsData\_id**),.**out**(**RsData\_ex**));**  Reg **#(.**width**(**32**))**RtData\_id\_ex**(.**clk**(**clk**),.**reset**(**Stall**),.**enable**(**1'b1**),.**in**(**RtData\_id**),.**out**(**RtData\_ex**));**  //--------------------------------------------------------------------------------------------------------  ////////////////////////////////【EX】///////////////////////////////  **wire** **[**31**:**0**]**ALUResult\_mem**;**  **wire** **[**1**:**0**]**ForwardA**,**ForwardB**;**  **wire** **[**4**:**0**]**RegWriteAddr\_mem**;**  **wire** RegWrite\_mem**;**  //Forwarding  Forwarding forward\_unit**(.**RegWrite\_wb**(**RegWrite\_wb**),.**RegWrite\_mem**(**RegWrite\_mem**),.**RegWriteAddr\_wb**(**RegWriteAddr\_wb**),.**RegWriteAddr\_mem**(**RegWriteAddr\_mem**),**  **.**RsAddr\_ex**(**RsAddr\_ex**),.**RtAddr\_ex**(**RtAddr\_ex**),.**ForwardA**(**ForwardA**),.**ForwardB**(**ForwardB**));**  //MUX  **wire** **[**31**:**0**]**ALUSrcA\_d\_in**,**ALUSrcA\_d**,**ALUSrcB\_d**,**MemWriteData\_ex**;**  Mux4 **#(.**width**(**32**))**ALUSrcA\_mux4**(.**sel**(**ForwardA**),.**in0**(**RsData\_ex**),.**in1**(**RegWriteData\_wb**),.**in2**(**ALUResult\_mem**),.**in3**(**0**),.**out**(**ALUSrcA\_d\_in**));**  **assign** ALUSrcA\_d**=(**ALUSrcA\_ex**==**1**)?** Sa\_ex **:** ALUSrcA\_d\_in**;**  Mux4 **#(.**width**(**32**))**ALUSrcB\_mux4**(.**sel**(**ForwardB**),.**in0**(**RtData\_ex**),.**in1**(**RegWriteData\_wb**),.**in2**(**ALUResult\_mem**),.**in3**(**0**),.**out**(**MemWriteData\_ex**));**  **assign** ALUSrcB\_d**=(**ALUSrcB\_ex**==**1**)?** Imm\_ex **:** MemWriteData\_ex**;**  **assign** RegWriteAddr\_ex**=(**RegDst\_ex**==**1**)?** RdAddr\_ex **:** RtAddr\_ex**;**  //ALU  ALU ALU\_unit**(.**ALUCode**(**ALUCode\_ex**),.**A**(**ALUSrcA\_d**),.**B**(**ALUSrcB\_d**),.**Result**(**ALUResult\_ex**));**  //-------------------------------【Ex/MEM】---------------------------  **wire** MemtoReg\_mem**,**MemWrite\_mem**;**  **wire** **[**31**:**0**]**MemWriteData\_mem**;**  **wire** **[**1**:**0**]**LwByte\_mem**;**  Reg **#(.**width**(**2**))**LwByte\_ex\_mem**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**(**LwByte\_ex**),.**out**(**LwByte\_mem**));**  Reg **#(.**width**(**3**))**Signal\_ex\_mem**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**({**MemtoReg\_ex**,**RegWrite\_ex**,**MemWrite\_ex**}),.**out**({**MemtoReg\_mem**,**RegWrite\_mem**,**MemWrite\_mem**}));**  Reg **#(.**width**(**32**))**ALUResult\_ex\_mem**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**(**ALUResult\_ex**),.**out**(**ALUResult\_mem**));**  Reg **#(.**width**(**32**))**MemWriteData\_ex\_mem**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**(**MemWriteData\_ex**),.**out**(**MemWriteData\_mem**));**  Reg **#(.**width**(**5**))**RegWriteAddr\_ex\_mem**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**(**RegWriteAddr\_ex**),.**out**(**RegWriteAddr\_mem**));**  //-----------------------------------------------------------------------------------------------------------------------  /////////////////////////////【MEM】//////////////////////////////////  //DataRam  **wire** **[**31**:**0**]**RamOut\_mem**;**  DataRam data\_unit**(.**clk**(**clk**),.**addr**(**ALUResult\_mem**),.**LwByte**(**LwByte\_mem**),.**din**(**MemWriteData\_mem**),.**we**(**MemWrite\_mem**),.**dout**(**RamOut\_mem**));**  //----------------------------------【MEM/WB】----------------------------------------------------------------------------------------  **wire** MemtoReg\_wb**;**  **wire** **[**31**:**0**]**ALUResult\_wb**;**  **wire** **[**31**:**0**]**RamOut\_wb**;**  Reg **#(.**width**(**2**))**WB\_mem\_ex**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**({**RegWrite\_mem**,**MemtoReg\_mem**}),.**out**({**RegWrite\_wb**,**MemtoReg\_wb**}));**  Reg **#(.**width**(**32**))**ALUResult\_mem\_wb**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**(**ALUResult\_mem**),.**out**(**ALUResult\_wb**));**  Reg **#(.**width**(**5**))**RegWriteAddr\_mem\_wb**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**(**RegWriteAddr\_mem**),.**out**(**RegWriteAddr\_wb**));**  Reg **#(.**width**(**32**))**RamOut\_mem\_wb**(.**clk**(**clk**),.**reset**(**1'b0**),.**enable**(**1'b1**),.**in**(**RamOut\_mem**),.**out**(**RamOut\_wb**));**  //-----------------------------------------------------------------------------------------------------------------------------------  ///////////////////////////【WB】/////////////////////////////////////  **assign** RegWriteData\_wb**=(**MemtoReg\_wb**==**1**)?** RamOut\_wb **:** ALUResult\_wb**;**  **endmodule** |