1. ID/EX段流水段寄存器

流水段寄存器的理论依据：同其他流水几个段寄存器一样，段寄存器的存在就是为了暂时性的存储数据，从上一个步骤跳到下一个步骤，在ID/EX段寄存器中，就是把译码那一段突破到计算这一段。前面的译码传入段寄存器很多信号，而且段寄存器的所接收的大部分的信号也是来源至此，除此之外，段寄存器中还由时钟接收到时钟信号，并且为了测试的方便已经增加代码思路的可读性，接收一个reset的信号。除此之外，还有cu\_stall和cu\_flush的信号传入，这两个信号分别代表着阻塞和冲刷。相信对流水线有过一定了解的就明白，流水线中不可避免的有冒险或者说是冲突，分为数据冒险和结构冒险。数据冒险和结构冒险主要是EX段负责所以不做累述，但一旦发生冒险，触发出cu\_stall或者cu\_flush这样的写使能的信号，就相当于优先级在先的控制这个流水段寄存器。剩下的就是接受信号，由于这次工程较大，所以信号较多。

//input:

Input clk;//时钟

Input reset;//重置

Input cu\_stall;//阻塞

Input cu\_flush;//冲刷

Input id\_rd\_addr[4:0];//id段译码出来的rd的地址

Input idex\_mem\_w;//在id段译码出来的控制存储器的写的使能

Input idex\_mem\_r;// 在id段译码出来的控制存储器的读的使能

Input idex\_reg\_w;// 在id段译码出来的控制寄存器的写的使能

Input idex\_branch;// 在id段译码出来的是否为branch指令判断

Input idex\_condition;// 在id段译码出来的ALU中选择器的将会选择的信号

Input idex\_of\_w\_disen;

Input idex\_exres\_sel;// 在id段译码出来的ex段的选择器的选择端口

Input idex\_B\_sel;// 在id段译码出来的B\_in 的选择；

Input idex\_ALU\_op[3:0] ;// 在id段译码出来的alu中的op端。

Input idex\_shamt\_sel;//桶形移位器的shift amount 的选项

Input idex\_shamt[4:0];//桶形移位器的5位shift amount

Input idex\_shift\_op[1:0];//桶形移位器的2位op，控制移位的种类的

Input idex\_imm\_ext[31:0];//被符号拓展了的数据。

Input idex\_rd\_addr[4:0];//寄存器rd的地址

Input idex\_pc[31:0];//32位的pc

Input idex\_pc\_4 [31:0];//PC+4,即通常情况下

Input idex\_load\_sel[2:0]//如果是load类型指令的话，根据各种不同的分工译码。

Input idex\_store\_sel[2:0]//如果是store类型指令的话，根据各种不同的分工译码。

Input idex\_op\_A[31:0];//

Input idex\_op\_B[31:0];//

Input idex\_rs\_addr[4:0];//寄存器rs的地址

Input idex\_rt\_addr[4:0];//寄存器rt的地址

Input idex\_cp0\_dst\_addr[4:0];//

而在输出的时候，基本上的信号都是一样的，就不做累述了。

写代码的时候，因为是寄存器，声明的时候是output reg \*\*\*\*\*[x:0];就可以实现寄存器的功能了。

考虑到阻塞和冲刷这两种情况所以分为两种情况写，写这个代码其实也就是代码长度比较长，很多地方的逻辑都是重复应用的，所以就举个例子

以 idex\_rs\_addr[4:0]为例。

Input idex\_rs\_addr[4:0];

Output reg exec\_rs\_addr[4:0];

always@(neglect clk)

if(!cu\_stall)

exec\_rs\_addr=idex\_rs\_addr;//不阻塞情况下直接给寄存器赋值

if(cu\_flush)

exec\_rc\_addr=0;//全部刷掉

其他的信号也大致一致，仿照即可。

1. 补分指令的测试

通过手写指令及汇编指令测试

Bgtz、Bgez、blez、bltz:

功能描述：

1. bgez

若寄存器rs的值大于等于0，则指令的转移地址由有符号偏移量左移2位来决定，即将offset左移两位之后，进行符号拓展并与当前PC相加。

1. bgtz

若寄存器rs的值大于0，则指令的转移地址由有符号偏移量左移2位来决定，即将offset左移两位之后，进行符号拓展并与当前PC相加。

1. blez

若寄存器rs的值小于等于0，则指令的转移地址由有符号偏移量左移2位来决定，即将offset左移两位之后，进行符号拓展并与当前PC相加。

1. bltz

若寄存器rs的值小于0，则指令的转移地址由有符号偏移量左移2位来决定，即将offset左移两位之后，进行符号拓展并与当前PC相加。

指令测试：

Bltz的测试：

首先把 4号、8号、16、27、28、29号寄存器的值分别置为 0 、1 、-1、0、0、0；

地址0 000001 00100 00000 (32)

操作码 寄存器地址 0 偏移量

地址4（store 指令 给 27 寄存器赋值3）

地址8 000001 01000 00000 (24)

操作码 寄存器地址 0 偏移量

地址12 （store 指令 给 28 寄存器赋值3）

地址16 000001 10000 00000 (16 )

操作码 寄存器地址 0 偏移量

地址20 （store 指令 给 29 寄存器赋值3）

地址24、28做死循环

地址32 （设计返回原地址的下一个目标地址）

查看寄存器的值 27 为0 ，28为0,29为3正确。

Blez的测试：

首先把 4号、8号、16、27、28、29号寄存器的值分别置为 0 、1 、-1、0、0、0；

地址0 000110 00100 00000 (32)

操作码 寄存器地址 0 偏移量

地址4（store 指令 给 27 寄存器赋值3）

地址8 000110 01000 00000 (24)

操作码 寄存器地址 0 偏移量

地址12 （store 指令 给 28 寄存器赋值3）

地址16 000110 10000 00000 (16 )

操作码 寄存器地址 0 偏移量

地址20 （store 指令 给 29 寄存器赋值3）

地址24、28做死循环

地址32 （设计返回原地址的下一个目标地址）

查看寄存器的值 27 为3 ，28为0,29为3正确。

Bgez的测试：

首先把 4号、8号、16、27、28、29号寄存器的值分别置为 0 、1 、-1、0、0、0；

地址0 000001 00100 00001 (32)

操作码 寄存器地址 1 偏移量

地址4（store 指令 给 27 寄存器赋值3）

地址8 000001 01000 00001 (24)

操作码 寄存器地址 1 偏移量

地址12 （store 指令 给 28 寄存器赋值3）

地址16 000001 10000 00001 (16 )

操作码 寄存器地址 1 偏移量

地址20 （store 指令 给 29 寄存器赋值3）

地址24、28做死循环

地址32 （设计返回原地址的下一个目标地址）

查看寄存器的值 27 为3 ，28为3,29为0正确。

Bgtz的测试：

首先把 4号、8号、16、27、28、29号寄存器的值分别置为 0 、1 、-1、0、0、0；

地址0 000111 00100 00000 (32)

操作码 寄存器地址 0 偏移量

地址4（store 指令 给 27 寄存器赋值3）

地址8 000111 01000 00000 (24)

操作码 寄存器地址 0 偏移量

地址12 （store 指令 给 28 寄存器赋值3）

地址16 000111 10000 00000 (16 )

操作码 寄存器地址 0 偏移量

地址20 （store 指令 给 29 寄存器赋值3）

地址24、28做死循环

地址32 （设计返回原地址的下一个目标地址）

查看寄存器的值 27 为0 ，28为3,29为0正确。