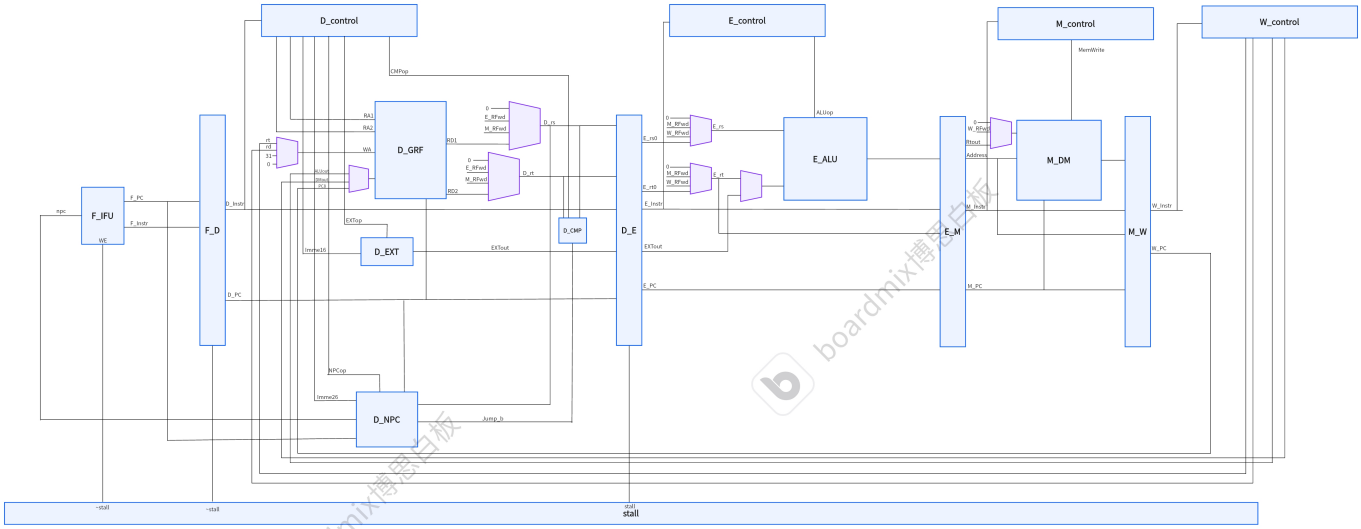
顶层设计



R型指令：

31:26	25:21	20:16	15:11	10:6	5:0
opcode	rs	rt	rd	shamt	func

I型指令：

31:26	25:21	20:16	15:0
opcode	rs(base)	rt	imme16(offset)

J型指令：

31:26	25:0
opcode	imme26(instr_index)

要实现的指令集：

指令类型	信号名	数量	指令
算术逻辑寄存器	cal_r	6	add,sub,and,ori,slt,sltu
算术逻辑立即数	cal_i	3	addi,andi,ori,lui
读内存	lord	3	lb,lh,lw
写内存	store	3	sb,sh,sw
两数比较分支	BRANCHE	2	beq,bne
仅写入跳转	jump_l	1	jal
仅读取跳转	jump_r	1	jr
乘除运算指令	md	4	mult,multu,div,divu

指令类型	信号名	数量	指令
存入乘除槽	mt	2	mthi,mtlo
乘除槽取出	mf	2	mfhi,mflo

命名规范：

1. 每个模块均使用大写字母命名，命名方式为模块所在层级+英文简称，比如E_ALU表示处在E级的ALU模块
2. 每个非流水线寄存器模块的端口信号均以大写字母开头，辅以小写字母，命名方式为英文简称（clk、reset和enable除外）,对于流水线寄存器中需要进行流水的信号，输入信号用前一个阶段+信号简称命名，输入信号用后一个阶段+信号简称命名，比如F_PC、D_PC
3. 在mips.v中所有实例化模块均以模块命名的小写字母形式命名，比如e_alu,所有连线信号均以小写字母命名，命名方式为英文简称，英文简称有冲突时辅以英文说明
4. 四个流水线寄存器使用前后层级+REG命名，比如D_E_REG

主要部件

1. PC

> 在P6中，IM脱离了CPU，成为了外部存储器，因此IFU缩减为PC，仅起到存储指令地址的寄存器的作用。

信号	方向	描述
clk	输入	时钟信号
reset	输入	同步复位信号
enable	输入	写使能信号
NPC[31:0]	输入	下一条指令的地址
PC[31:0]	输出	当前指令的地址

2. GRF

> 与P5相比没有变化

信号	方向	描述
clk	输入	时钟信号
reset	输入	同步复位信号
RegWrite	输入	控制信号，为真时GRF方可写入数据

信号	方向	描述
RA1[4:0]	输入	读地址
RA2[4:0]	输入	读地址
WA[4:0]	输入	写地址
WD[31:0]	输入	要写入的数据
PC[31:0]	输入	当前指令的地址
RD1[31:0]	输出	读出的值
RD2[31:0]	输出	读出的值

3. CMP

> 新增控制信号来决定使用哪种比较方式，新增bne指令的比较

信号	方向	描述
Rtout[31:0]	输入	参与比较的第一个数
Rsout[31:0]	输入	参与比较的第二个数
CMPop[1:0]	输入	比较控制信号
Jump_b	输出	branch类指令是否跳转信号

4. EXT

> 同P5

信号	方向	描述
Imme16[15:0]	输入	待扩展的16位立即数
EXTop[1:0]	输入	扩展方式控制信号
Imme32[31:0]	输出	扩展之后的32位立即数

5. NPC

> 同P5

信号	方向	描述

信号	方向	描述
Offset[15:0]	输入	16位立即数
Instr_index[25:0]	输入	jal指令相关立即数
NPCop[2:0]	输入	跳转方式选择信号
JrReg[31:0]	输入	jr指令目的寄存器
F_PC[31:0]	输入	F级PC值
D_PC[31:0]	输入	D级PC值
Zero	输入	判断branch类指令是否跳转
Npc[31:0]	输出	下一条指令的地址

6. ALU

> 增加运算方式，其余同P5

信号	方向	描述
A[31:0]	输入	第一个运算数
B[31:0]	输入	第二个运算数
ALUop[2:0]	输入	ALU的控制信号
ALUout[31:0]	输出	ALU的计算结果

7. MDU

> P6新增模块，用于实现乘除法指令

信号	方向	描述
clk	输入	时钟信号
reset	输入	同步复位信号
Start	输入	开始乘除指令的信号
MDUop[2:0]	输入	乘除指令控制信号
Rs_data[31:0]	输入	参与运算的第一个数
Rt_data[31:0]	输入	参与运算的第二个数
Busy	输出	当前是否在进行乘除指令，为1则正在进行

信号	方向	描述
HI	输出	储存乘运算的高32位和除运算的余数
LO	输出	储存乘运算的低32位和除运算的商

8. BE

信号	方向	描述
Addr[31:0]	输入	写入内存的地址
BEop[2:0]	输入	存储指令的控制信号
Rt_data[31:0]	输入	待修改的数据
Byteen[3:0]	输出	字节使能信号
WD[31:0]	输出	经过存储指令处理后的要写入内存的数据

9. DE

信号	方向	描述
A[31:0]	输入	写入内存的地址
Din[31:0]	输入	带要进行扩展的数据
DEop[2:0]	输入	数据扩展控制信号
Dout[31:0]	输出	扩展之后的数据

控制器设计

> 依然采用分布式译码，同P5相比，删除了部分信号，新增了部分信号，同时为了方便课上进行添加，信号的位数都比实际所需位数高1位

信号	方向	描述
Instr[31:0]	输入	输入的32位指令信号
Rs[4:0]	输出	
Rt[4:0]	输出	
Rd[4:0]	输出	add,sub指令的目的寄存器
Shamt[4:0]	输出	
Offset[15:0]	输出	
Instr_index[25:0]	输出	

信号	方向	描述
ALUout[3:0]	输出	ALU的控制信号
EXTop[1:0]	输出	EXT的控制信号，1时符号扩展，反之进行0扩展
RegWrite	输出	GRF写使能信号，为1时方可写寄存器，具体实现时恒为1
NPCop[2:0]	输出	NPC控制信号
CMPop[1:0]	输出	CMP控制信号
ALUSrc[2:0]	输出	选择ALU第二个数据的来源
BEop[2:0]	输出	BE模块控制信号
DEop[2:0]	输出	DE模块控制信号
MDUop[3:0]	输出	MDU模块控制信号
Start	输出	乘除模块启动信号，为1时开始进行乘除运算
FwSel[2:0]	输出	选择转发数据
RegWriteDst[4:0]	输出	每个指令写寄存器的地址
Tuse_rs[1:0]	输出	每个指令rs寄存器的Tuse
Tuse_rt[1:0]	输出	每个指令rt寄存器的Tuse
Tnew_E[1:0]	输出	每个指令在E级的Tuse
Tnew_M[1:0]	输出	每个指令在M级的Tuse

冲突解决

> 利用AT法解决
> Tuse是指某一指令在D级再经过多少个时钟周期就必须要使用相应的数据，Tnew是指位于某个流水级的某个指令，它经过多少个时钟周期可以算出结果并且存储到流水线寄存器里。

指令	A信号	$Tnew-E$	$Tnew-M$	$Tuse-rs$	$Tuse-rt$
cal_r	rd	1	0	1	1
cal_i	rt	1	0	1	null
lord	rt	2	1	1	null
store	0	0	0	1	2
branch	0	0	0	0	0
jump_l	31	0	0	null	null
jump_r	0	0	0	0	null

指令	A信号	$Tnew-E$	$Tnew-M$	$Tuse-rs$	$Tuse-rt$
md	0	0	0	1	1
mf	rd	1	0	null	null
mt	0	0	0	1	null

> A信号是该指令的目的寄存器，如没有则取0;null表示该指令不使用该寄存器，在具体实现时信号用可取的
> 最大值表示。

阻塞设计

> 设计专门的stall模块来处理合适要进行阻塞，stall模块会输出stall信号

首先要明确阻塞发生的条件：当冲突发生时，较旧的指令无法及时提供给较新的指令其所计算出来的值，便需要插入一个nop空泡，也就是使用阻塞，具体条件是**旧指令寄存器的写入地址和新指令寄存器的读出地址相同且读出地址不为0，而且旧指令的$Tnew$ $>$ 新指令的$Tuse$。**

课程要求**阻塞发生在D级**，因此当进行阻塞时，要将PC冻结，将F_D_REG的值冻结，将D_E_REG的值清零，也就是说我们要为IFU模块增加使能信号，为F_D_REG增加使能信号，为D_E_REG增加复位信号（异步复位），但是为了模块的统一性，我们为所有流水线寄存器都设置使能端enable和复位端reset,对于非上述两个流水线寄存器，使能信号恒为1，复位信号恒为0。

当进行乘除运算时，要求乘法进行5个时钟周期，除法进行10个时钟周期，但是实际上MDU的结果在一个周期内就可以得到，因此需要对乘除指令进行额外的阻塞判断。当Busy或者Start为1时，说明当前正在进行乘除运算，因此我们需要进行阻塞，此时如果在D级出现了和乘除槽有关的指令我们都需要进行阻塞，具体实现如下：

```
    wire stall;
    wire stall_mdu;
    wire STALL = stall | (stall_mdu && (D_mt | D_mf | D_md));
```

其中stall_mdu是MDU模块的输出，表明此时Busy和Start中有一个为1，而D_mt、D_mf、D_md说明D级是和乘除槽有关的指令。

转发设计

和阻塞一样，首先要明确转发发生的条件。当一个较新的指令要使用的寄存器里的值还没有被较旧的指令写入到寄存器里面，但是已经被较旧的指令给算出来，这时候较新的指令便可以直接使用较旧的指令计算出来的值而不是使用寄存器里的值，这就是转发。

如何实现转发?

正如上文所说，所谓转发，就是新指令直接使用旧指令的结果，因此只需要比较新旧指令的目的寄存器的地址是否相同便可，相同而且$Tuse>=Tnew$便可以转发，否则便阻塞。 对于P6的指令集，写入寄存器的来源有四个：jal的PC+8，ALU的输出结果，DM的输出结果、MDU的输出结果。而对于E级的转发只能是PC+8，因为这时候ALUout和DMout还没有产生，M级的转发有PC+8、ALUout和MDUout，W级的转发有PC+8、ALUout、DMout和MDUout,我们可以在控制器中增加转发结果的选择信号，依照指令来选择转发结果，比如add就选择ALUout，lw就选择DMout，具体数据通路如下：

```
需求者：D级GRF的两个输出端口，E级ALU的两个输入端口，M级DM的输入端口。
供给者：E级的PC+8，M级的PC+8、ALUout和MDUout,W级的PC+8、ALUout、DMout和MDUout
转发数据流：E_RFwd -> D_RD1
           E_RFwd -> D_RD2
           M_RFwd -> D_RD1
           M_RFwd -> D_RD2
           W_RFwd -> D_RD1      //GRF内部转发
           W_RFwd -> D_RD2      //GRF内部转发
           M_RFwd -> E_ALUinA
           M_RFwd -> E_ALUinB
           W_RFwd -> E_ALUinA
           W_RFwd -> E_ALUinB
           W_RFwd -> M_DMin
```

延迟槽

清空延迟槽 在默认情况下，延迟槽中的指令都是默认执行的，但是有些指令（基本上都是分支跳转指令）在某些条件下不想让延迟槽中的指令执行，这就是清空延迟槽，比如bltzall，bgezall。

如何实现清空延迟槽?

法1：在CMP模块中新增一个输出flush，当判断为不需要进行跳转时，将flush置为1，并将F_D_REG清空，但是需要特别注意的是，当且仅当当前周期的阻塞信号为0时，我们才能执行清空延迟槽的操作。

省流：当且仅当当前周期的阻塞信号为0且CMP判断为需要清空延迟槽时，将F_D_REG清空。

具体如下：

```
//D_CMP.v
flush = (CMPop == `CMPop_new) && (!jump_b)

//mips.v
F_D_REG_reset = (~STALL && flush);
```

法2：当监测到该指令不跳转时，F级的指令插入nop,具体如下：

```
F_instr = (!D_b_jump && D_bgezall) ? 32'd0 : im[pc[13:2] - 12'hc00];
```

条件储存

对于条件储存的指令，基本思想是统一在W级进行处理，当在W级之前出现要读条件储存指令的指令时，都要将它阻塞在D级。 因此要对阻塞模块进行修改，在stall条件判断的读写寄存器是否相等的地方增加对E、M级指令的判断，比如`lwmx`指令,其要写的目的寄存器是4号寄存器和5号寄存器， 因此将stall模块修改为：

```
    wire stall_E_rs = ((D_RegReadDst_rs != 5'b00_000) && ((E_lwmx)?
(D_RegReadDst_rs == 5'b4 || D_RegReadDst_rs == 5'b5):(D_RegReadDst_rs ==
E_RegWriteDst)) && (Tnew_E > Tuse_rs));
    wire stall_E_rt = ((D_RegReadDst_rt != 5'b00_000) && ((E_lwmx)?
(D_RegReadDst_rt == 5'b4 || D_RegReadDst_rt == 5'b5):(D_RegReadDst_rt ==
E_RegWriteDst)) && (Tnew_E > Tuse_rt));
    wire stall_M_rs = ((D_RegReadDst_rs != 5'b00_000) && ((M_lwmx)?
(D_RegReadDst_rs == 5'b4 || D_RegReadDst_rs == 5'b5):(D_RegReadDst_rs ==
M_RegWriteDst)) && (Tnew_M > Tuse_rs));
    wire stall_M_rt = ((D_RegReadDst_rt != 5'b00_000) && ((M_lwmx)?
(D_RegReadDst_rt == 5'b4 || D_RegReadDst_rt == 5'b5):(D_RegReadDst_rt ==
M_RegWriteDst)) && (Tnew_M > Tuse_rt));
    wire stall_E = stall_E_rs | stall_E_rt;
    wire stall_M = stall_M_rs | stall_M_rt;
    assign stall = stall_E | stall_M;
```

对于具体的条件判断，可以在`mips.v`模块中实现，而不是在其他模块中实现。这个条件由于仅仅在W级进行判断，所以在W级之前的control会出现不定值`x`，因此要使用三等号`===`和`!==`来进行判断。 对于不是进行条件判断，而是直接确定写的寄存器的指令，也可以在`mips.v`中确定，然后将寄存器号直接连入W级的control。

思考题

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

之所以要将乘除法部件单拎出来，是因为乘除法指令相比于ALU中的其他计算指令效率低很多，如果将MDU模块和ALU模块合并则会显著降低CPU的效率。
之所以采用HI和LO两个寄存器一是为了避免结果超出单个寄存器的宽度，有些乘法的计算结果可能超过了
32位，同时除法运算通常需要同时存储商和余数；二是为了提高性能，两个独立的寄存器能够允许程序在
需要时快速访问结果，而不需要通过复杂的指令来拼接和操作寄存器中的数据。

2. 真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

对于乘法：CPU使用硬件电路（如加法器和移位器）完成二进制乘法，高性能CPU采用阵列乘法器或树形乘法器，快速生成和累加部分积，流水线将乘法分成多个步骤，逐步完成，提高效率。

对于除法，使用移位减法或迭代逼近算法完成二进制除法，流水线分解除法步骤，逐步计算商和余数，浮点除法可能通过反复乘法逼近结果。

将乘法和除法分成多个阶段，让多个指令同时处理不同阶段，提升吞吐量，除法比乘法慢，因为操作更复杂，需要更多步骤。

3. 请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的?

因为乘法的执行需要5个周期，除法的执行需要10个周期，但是实际上乘除法的计算是在一个周期内就可以完成的，计算完成后为了满足周期数要求就要进行阻塞，在Start信号为1时开始计算，之后Start置
0，Busy信号置1，时钟周期数满足时候Busy置0.因此我将stall_mdu信号作为MDU模块的输出，除了其他指令
的正常阻塞之外，当Start或Busy为1时也要进行阻塞，因此stall_mdu = Busy | Start,所以最终的阻塞信号STALL = stal| Busy
| Start。

4. 请问采用字节使能信号的方式处理写指令有什么好处？ (提示：从清晰性、统一性等角度考虑)

在清晰性上，byteen的值能够清楚的表示写入数据的位置；
在统一性上，|byteen为1即表示需要写入数据，相当于将存储器的使能信号并入byteen中。

5. 请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢?

不是同一字节，在中间过程中进行了移位操作。
当要取出的数据是1字节时，按字节读和按字节写的效率更高，因为按字读和按字写还要进行取位操作。

6. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助?

将指令进行分类，分类方式上面已经给出，同类的指令具有相同的性质，比如具有相同的Tuse和Tnew，
但是也有例外的发生，比如addi是符号扩展，而立即数运算的其他指令都是零扩展，对于这种情况只需
要特殊情况特殊处理就行，也就是将addi单拎出来进行符号扩展,这样做能明显减少control部分的码量，
提高了可读性，降低了复杂度。

7. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的?

mf型的指令和cal_r型的指令等。解决方法是进行转发，将E级产生的乘除法计算结果转发到D级。

```
mfhi $3
add $2,$3,$4
```

8. 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果

采用的是手动构造的样例。构造策略如下：
相比于P5，转发通路几乎没有变化，因为乘除法的计算结果可以看作是ALU的计算结果。对于进行测试的数据，
我首先考虑的是较大的数据，或者说是边界值，来检测乘法是否满足高位储存在HI，低位是否储存在LO。