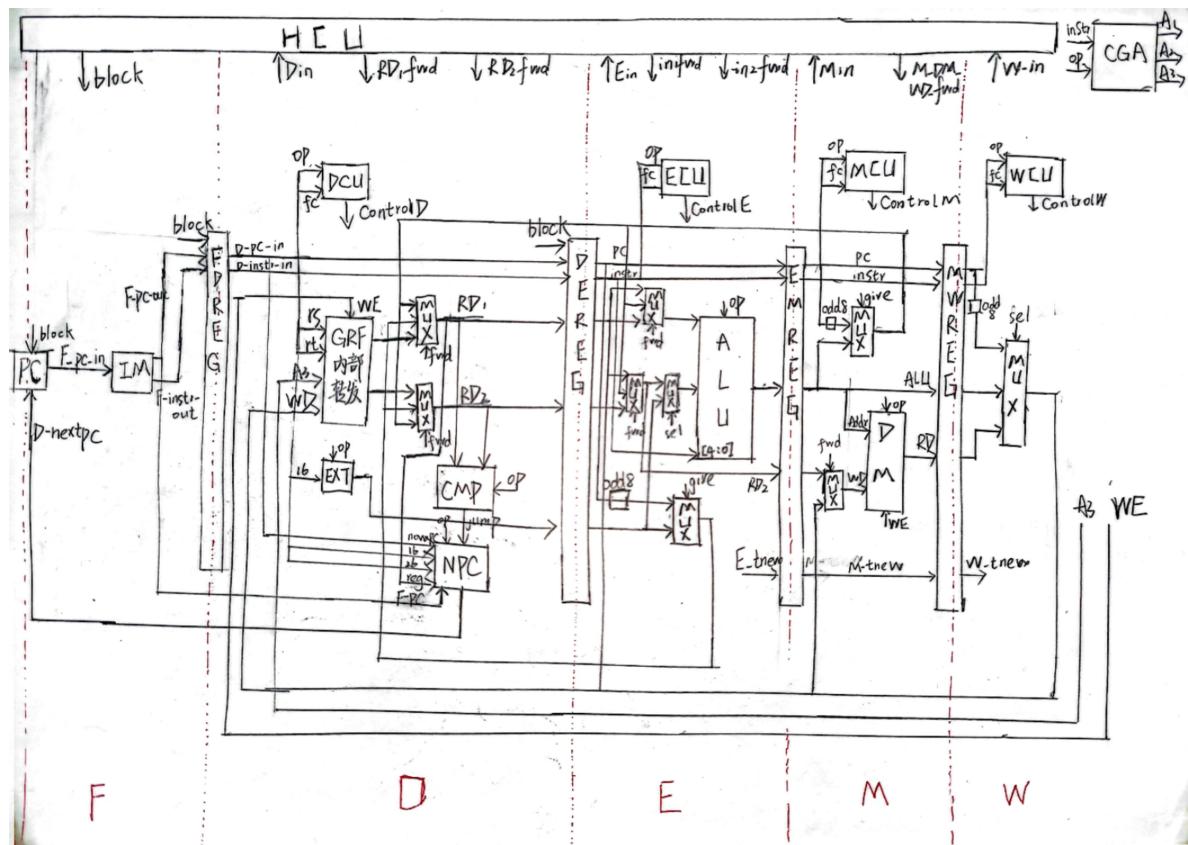


设计草稿

顶层电路

- 该电路为p5电路，p6与其差异主要有如下几点：
 - IM、与DM均外置，mips.v的输出端口增加了*i_inst_rdate, i_inst_addr, m_date_rdate, m_date_addr, m_date_wdate, m_date_byteen, m_ins_t_addr, w_grf_we, w_grf_addr, w_grf_wdata, w_inst_addr*。
 - DM扩展了许多，使用always@(*)统一处理DM的输入和输出。
 - 新增了乘除槽，与ALU并立，in1与in2与ALU的in1,in2一致，输出数据往后流水。
 - 扩展了许多指令，扩展了DECODER、HCU、ALU、CMP。



指令分类

- 目前已有指令的分类

| 类型名 | 指令名 |
|---------|------------------------------|
| cal_rr | add, sub, and, or, slt, sltu |
| cal_ri | addi, andi(符号扩展), ori |
| load | lw, lh, lb |
| store | sw, sh, sb |
| md_cal | mult, multu, div, divu |
| md_read | mflo, mfhi |

| 类型名 | 指令名 |
|----------|-----------|
| md_write | mtlo,mthi |
| branch | beq,bne |
| shift | sll |
| lui | lui |
| jal | jal |
| jr | jr |
| j | j |

• 精简指令集的分类

指令类即具有实现相似性的若干指令，例如对于 MIPS-C3 指令集（包含了 P5、P6 课下要求的指令集），可以进行如下分类：

- 寄存器立即数计算： addi, addiu, slti, sltiu, andi, ori, xori, sll, srl, sra
- 寄存器寄存器计算： add, addu, sub, subu,slt, sltu, and, or, nor, xor, sllv, srlv, sra
- 根据寄存器分支： beq, bne, bgez, bgtz, blez, bltz
- 写内存： sw, sh, sb
- 读内存： lw, lh, lhu, lb, lbu
- 读乘除法寄存器： mfhi, mflo
- 写乘除法寄存器： mthi, mtlo, mult, multu, div, divu
- 跳转并链接： jal, jalr
- 跳转寄存器： jr, jalr
- 加载高位： lui
- 空指令： nop

上面的指令中并没有无条件跳转指令 j，这是因为 j 指令不产生数据也不使用数据，已被移出群

课上处理方法

仔细看RTL语言！！！

特别注意 \$signed！！！

计算类

- 一般与乘除部件有关，在DECODER中可以参考mult
- 应特别注意 \$signed()

对于madd指令，正确的写法为

```

//手动扩展到六十四位
{tHI,tLO} <= {HI,LO} + $signed({{32{in1[31]}}, in1[31:0]} *
$signed({{32{in2[31]}},in2[31:0]}));
//加一个64位的0
{tHI,tLO} <= {HI,LO} + $signed($signed(64'd0) + $signed(in1) *
$signed(in2));

```

跳转类

- 是否是条件跳转

如果是条件跳转，仅需在**CMP**中进行判断，然后再修改NPC。

- 是否是条件链接

- 如果是非条件链接，同jal，注意需要修改一下A3。
- 一般grf_write统一置为1'b1,如果是条件链接，那么应当根据jump信号判断A3的值，将jump、check(新增指令标示)向后流水，jump为1'b1时A3为目标寄存器，否则为0号寄存器。

- 是否需要在不跳转时清空延迟槽

如果题目要求在不跳转时清空延迟槽，那么需要在CMP中新增clear信号，然后根据clear信号与block信号确定是否要清空FD_REG

```

if (reset == `TRUE || (!block && clear)) begin
    D_instr <= 32'h0000_0000;
    D_pc <= `INITPC;
end

```

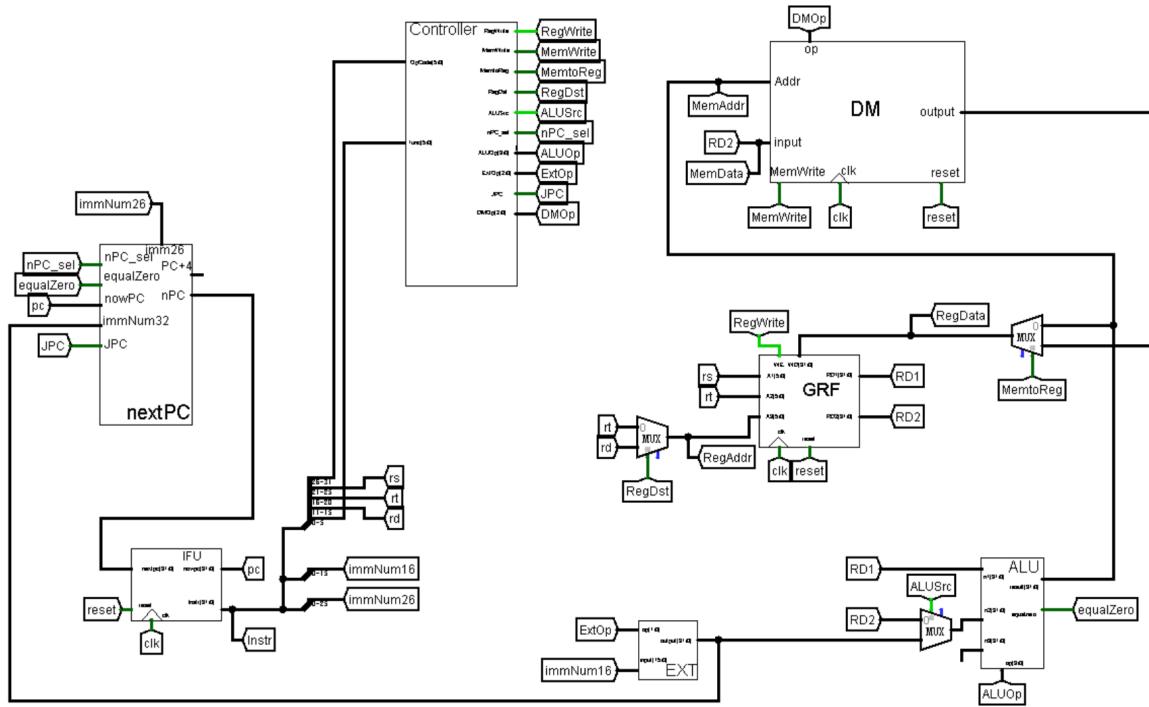
访存类

- 特点：在M级从DM中取出值后才知道要往哪个寄存器写

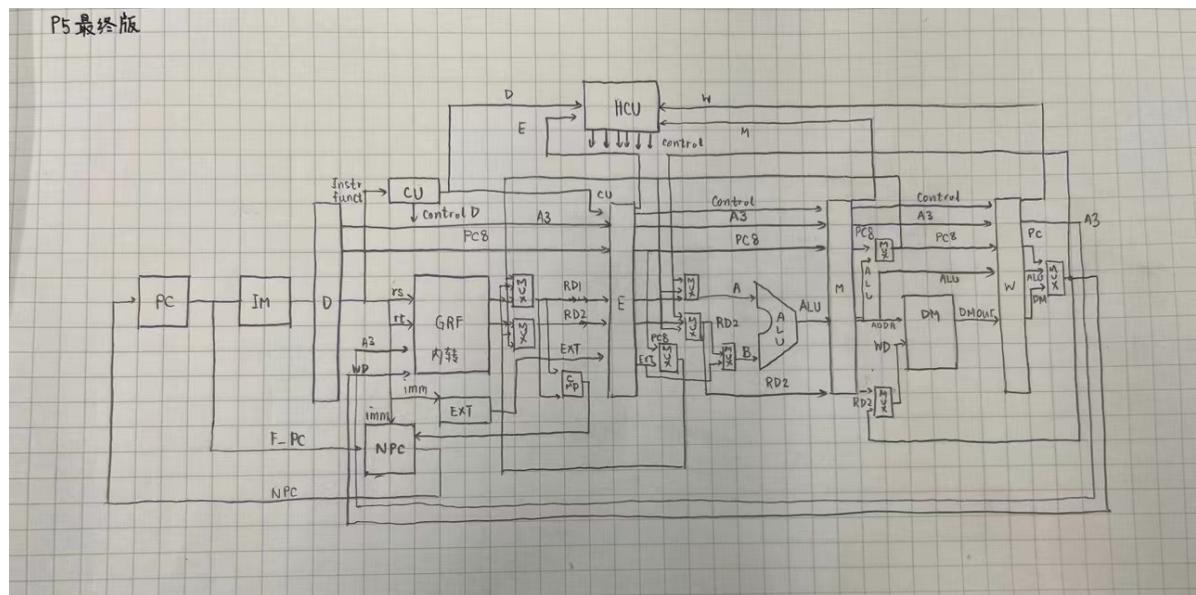
- 方法：

- 一般grf_write一般统一置为1'b1,如果D级的指令要读寄存器，而且后面的新指令可能要写这个寄存器，那么就block,具体方法是将HCU中判断A1等于A3处修改为一个三目运算。
- 还需要在M级根据DM中取出的值修改A3，这时还需要将flag(是否满足条件)、check(新增指令标示)向W流水以修改W的A3。

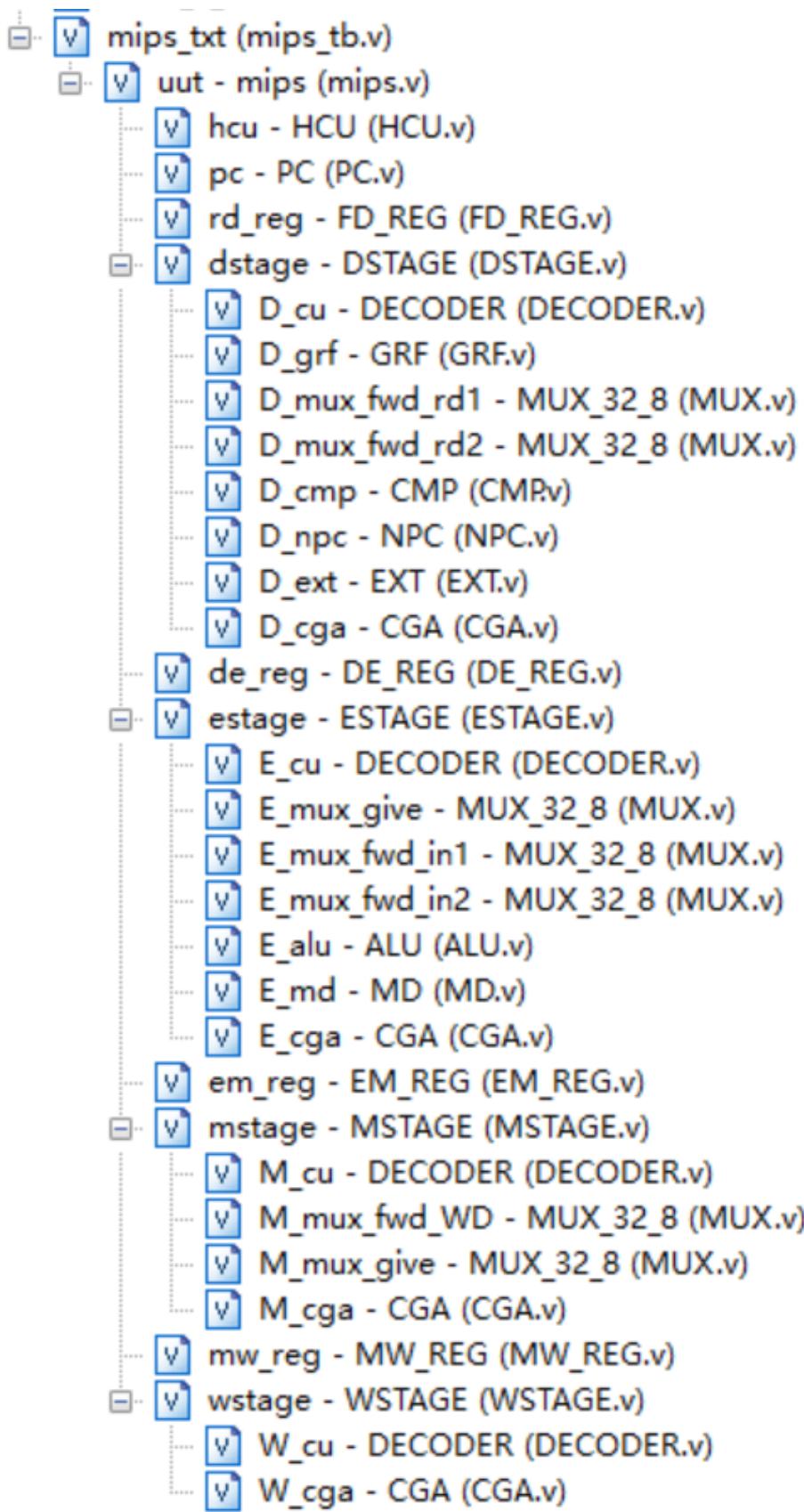
单周期Logisim顶层电路图



参照顶层电路图



电路架构



测试方案

- 手动构造测试数据重点测试乘除指令以及load、store指令
- 使用实验教程提供的测试代码进行初步测试
- 使用Python简单写了一个自动生成MIPS测试代码的代码，进行大范围的随机测试，以排查出其他指令的细节错误。

思考题

第一题 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？

- 在实际的CPU中，大部分在ALU中运算的操作都可以在一个周期内完成，而乘除法需要消耗长得多的时间，如果将乘除法整合进入ALU内，那么要么选择增长时钟周期，要么选择阻塞之后的指令，这都会造成极大的效率上的损失，与CPU的设计初衷不符合。
- HI与LO寄存器只与mult、multu、div、divu、mflo、mfhi、mtlo、mthi有关，其他指令均无法访问或者改变HI与LO内的值，即它们不属于通用寄存器，根据CPU设计高内聚低耦合的设计原则，应该将这两个寄存器内置在乘除法单元，而不是放在GRF内。

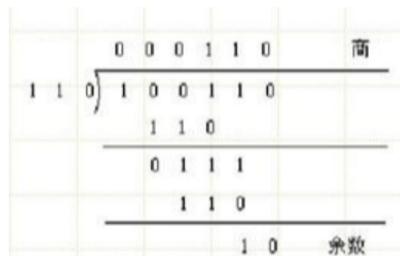
第二题 真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

在真实的流水线CPU中，乘除法是通过在每个周期计算少数几位结果，最终得到正确的答案。

- 对于乘法，每个周期计算特定的几位，最后累加求和得到正确的结果。
- 对于除法，通常是使用试商法最终得到正确的结果。

试商法解释：

设被除数dividend为A，除数divisor为B，位宽分别为M、N。二进制除法^①实际上和十进制的除法计算过程一样，下图是100110除110的例子。



而这种算法本质上是从被除数的第一位开始，截取宽度N的部分Q，例子中计算第一位是 $Q = 100$ 。

判断Q是否比除数B的某一倍数大，取最大的倍数作为当前位的商。二进制除法即是判断Q是否比B大，如果比B大，那么商这一位是1，并取Q-B的结果作为下一次判定的Q。

如果Q小于B，那么这一位商结果是0，同时Q按顺序再取一位，得到N+1宽度的Q，再次进行判定。依据这个算法，我们很容易写出一个基本的除法器。

但是这种算法要求我们知道除数B的实际位数，即不为0的最高位是多少。显然现实生活中我们只知道输入B是一个32位、8位的数，无法直接判断不为0的最高位。当然，也可以通过前置的电路首先进行判断，再将结果作为输入给到这种除法器。

第三题 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？

- 当D级流水线中的指令为乘除指令并且busy信号或者start信号为高电平时即决定阻塞

第四题 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）

- 要写哪个字节十分的清晰
- 可以省去写使能信号，变得更加简洁

第五题 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个字节？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？

- 在按字节读写时，实际从DM获得的数据和向DM写入的数据不一定是一个字节。如果是lw就需要读多个字节，如果是sw就需要存多次。
- 如果按字读写的话，而指令时sh, sb，那么就必须先读出字的值才能再写入，延长了数据通路，降低了效率。

第六题 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？

- 根据指令的特点对指令进行分类，这样在写或逻辑时就比较简单。
- 将各级流水的组合逻辑封装成单独的模块，减少了复杂性。
- 善用注释，主要有用//来区分不同的逻辑部分以及写注释来写变量的用处。

第七题 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？

- 第一类冲突是常规指令的冲突，这里和p5的冲突类型相同，将指令分成几种类别，通过一般的AT法即可解决。
- 第二类冲突是乘除法指令的冲突，需要将start、busy信号传送到HCU中来判断是否需要阻塞。

测试样例如下：

```
ori $t1,3
ori $t2,2
mtlo $t1
mthi $t2
mult $t1,$t2
mfhi $t3
mflo $t4
mult $t2,$t4
multu $t1,$t4
mfhi $t3
mflo $t4
div $t2,$t4
divu $t4,$t1
mfhi $t3
mflo $t4
nop
label:
beq $0,$0,label
```

第八题 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。

- 手动构造样例主要应该考虑几种能代表不同阻塞以及转发以及乘除的指令。
- 生成测试样例时的策略：
 - 检测内存是否对齐
 - 限制立即数的范围合法
 - 设立跳转区
 - 增加生成边界立即数(0,1, -1,65535.....)的概率值