厦门大学

《计算机组成原理》

课程实验报告

姓名：黄子安

学院：信息学院

系：软件工程系

专业：软件工程

学号：22920212204396

第六次实验　流水线 MIPS 处理器实验

1. 实验环境

- MIPS 汇编仿真器
- Windows 系统或 Mac os 环境下运行 Logisim 软件

2. 实验目的

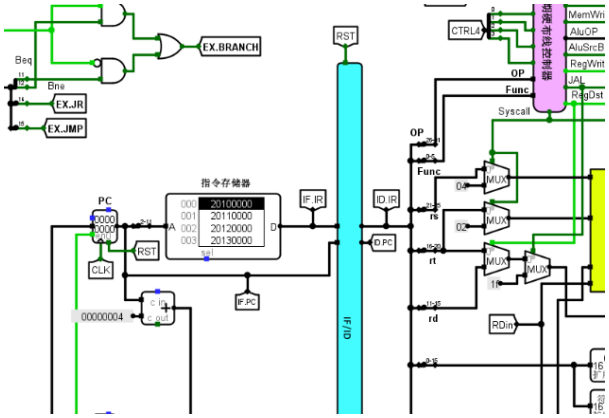
- 了解单周期 MIPS 处理器理想流水线数据通路的特点；
- 了解气泡流水线 MIPS 处理器数据通路处理控制相关和数据相关的工作原理；
- 了解重定向流水线 MIPS 处理器数据通路处理数据相关的特点以及与气泡流水线的区别；
- 理解动态分支预测流水线 MIPS 处理器解决控制相关的基本原理。

3. 实验内容

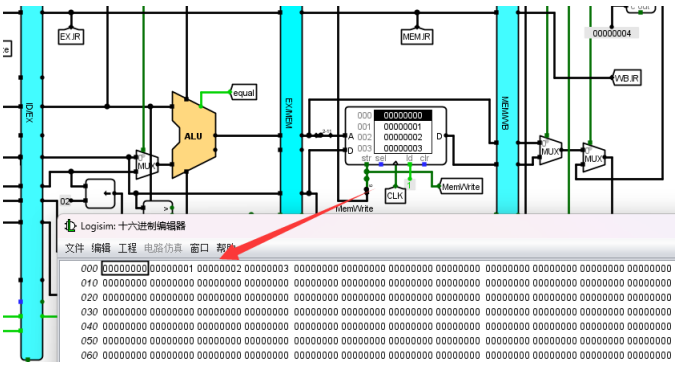
3.1 理想流水线 MIPS 处理器

3.1.1 基于理想流水线 MIPS 处理器数据通路运行测试程序 test.hex

在指令存储器中装载对应的机器码，之后启动时钟，点击总复位后程序开始运行



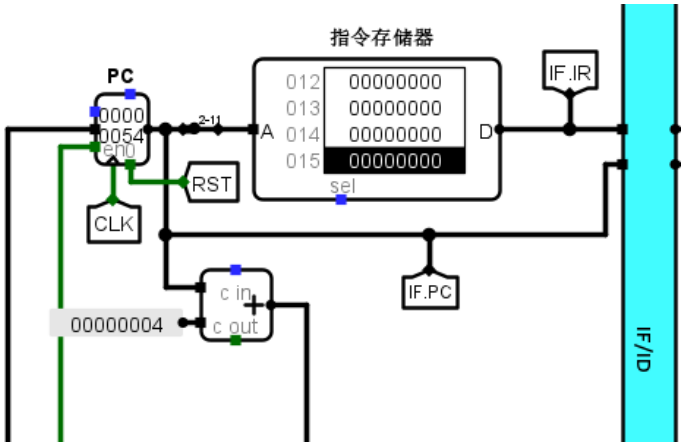
运行后可以看到程序将 0、1、2、3 分别存储到存储器的 0、1、2、3 号单元中



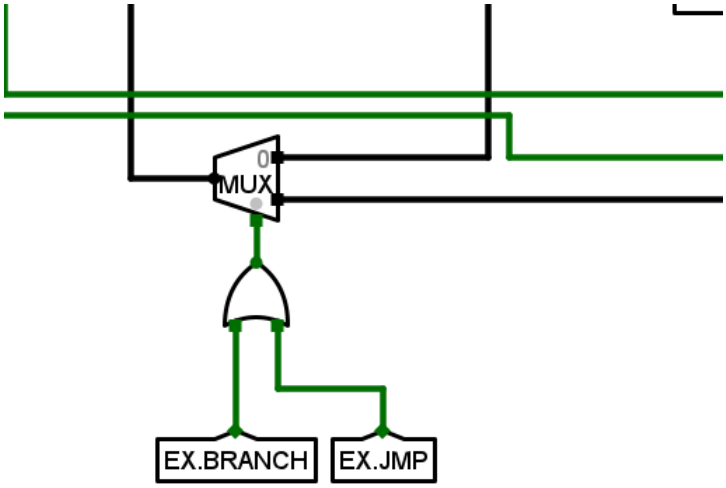
3.1.2 分析理想流水线 MIPS 处理器的数据通路

数据通路分析按照五个流水线的阶段进行

第一阶段：取指 IF

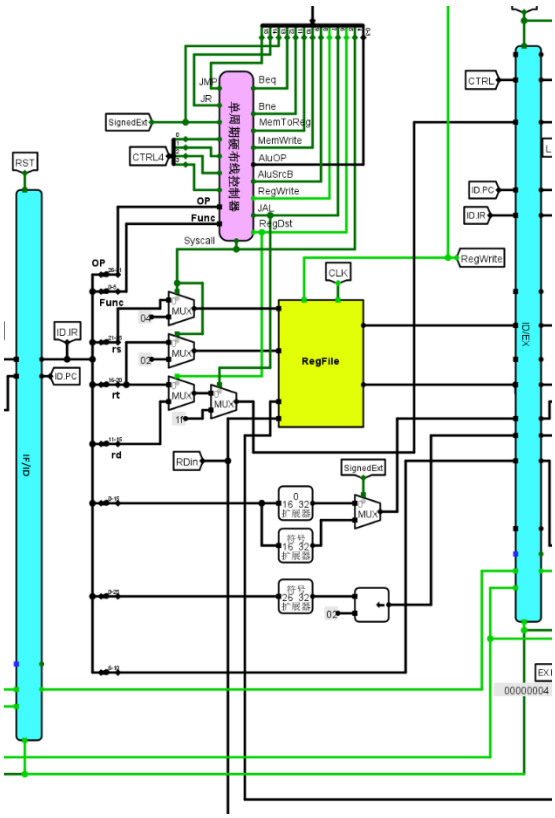


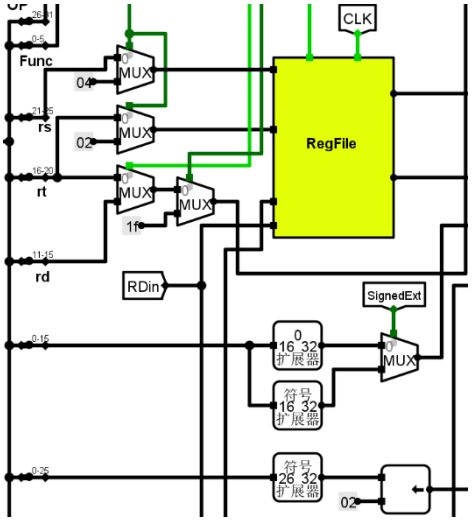
取指部分主要包括一个 PC 寄存器和一个指令寄存器，该部分在读取到指令后会在时钟的上跳沿将指令和指令对应的地址写入到 IF/ID 流水线中等待下一个阶段的程序使用，同时这里自带一个加法器，会自动计算 PC+4，如果是顺序执行会在下一个时钟的上跳沿将数据写入 PC 寄存器从而改变 PC 的值完成跳转



PC 的输入除了 PC+4 还有对应的跳转地址，会在后面阶段计算得到并根据对应的控制信号来决定是否输入 PC，该部分放到后面进行说明。

第二阶段：译码 ID



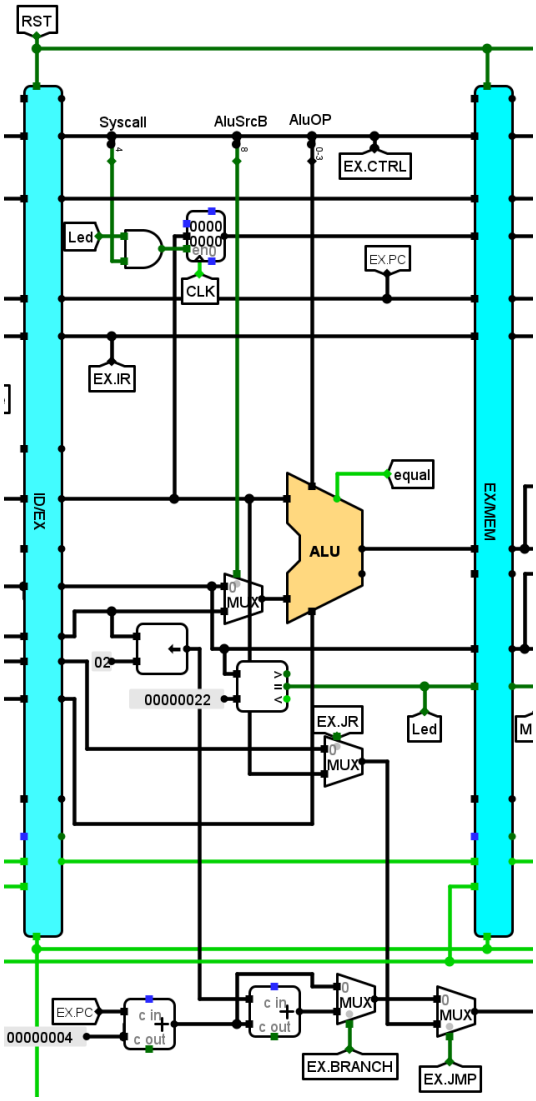


译码阶段会从 IF/ID 流水寄存器中读取对应的指令，之后使用分线器提取对应部分的数据，如果是寄存器将会进行寄存器寻址，寄存器堆的读取不需要时钟信号控制，对于给出的寄存器编号可以直接在输出引脚输出对应的寄存器值；如果是立即数则会进行符号扩展之后传送到 ID/EX 流水寄存器

控制器采用硬布线结构，不需要时钟驱动，当给出指令信号便能给出所有的控制信号(实际上还要考虑电信号在电路中的传输延迟)，无需时钟信号，所以译码阶段所有的输出结果（立即数、寄存器里的对应内容、控制信号）在第二个时钟周期到来前其实已经给出，因为读取第一个流水寄存器不需要时钟控制，但译码的所有结果不会写入 ID/EX 流水寄存器，只有当第二个时钟来临时才会完成写入，其余的流水寄存器也是同理。

这里还有几个特殊常量和 MUX 需要说明，04 和上面的 02 常量对应的多路选择器由 Syscall 控制信号控制。对于 Syscall 指令，rs 寄存器用于表示系统调用号，代表需要执行的系统调用的类型，而 rt 寄存器则用于传递系统调用的参数，这里 4 表示停机指令。1f 对应寄存器是 MIPS 寄存器堆的第 31 个寄存器，该寄存器用于保存子程序返回地址，从而完成子程序的调用。最后下面的 02 是为了实现 j 指令，该指令会获取 PC<<2 的低 28 位。

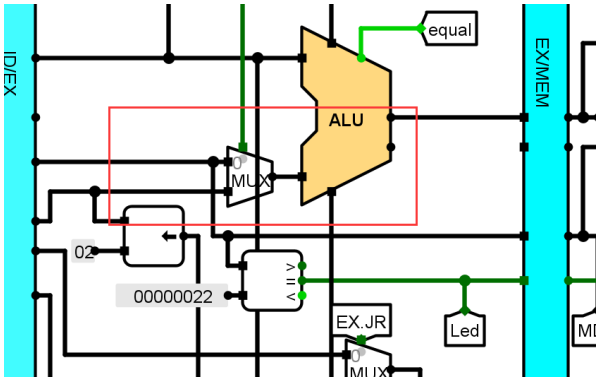
第三阶段：执行 EX



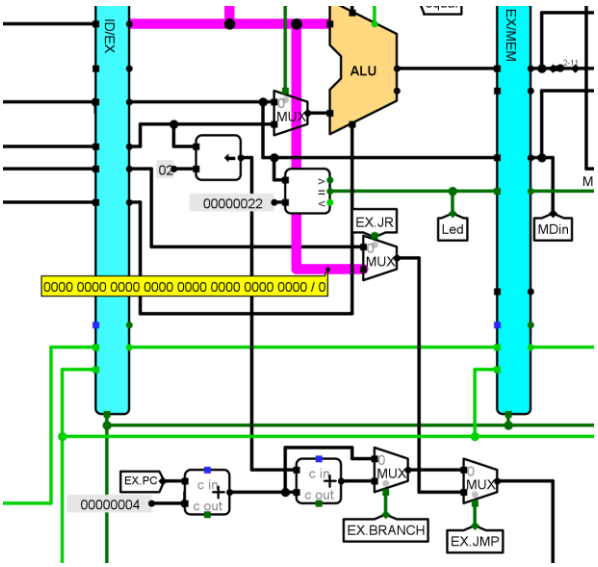
执行阶段主要用于完成计算，会从对应的总控制信号中获取需要的控制信号，之后使用运算器完成对应结果的计算

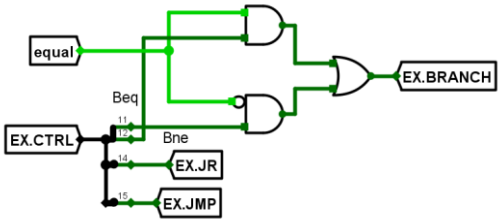
接下来分析多路选择器的作用

该多路选择器选择进入 ALU 的操作数是立即数还是寄存器中的数据



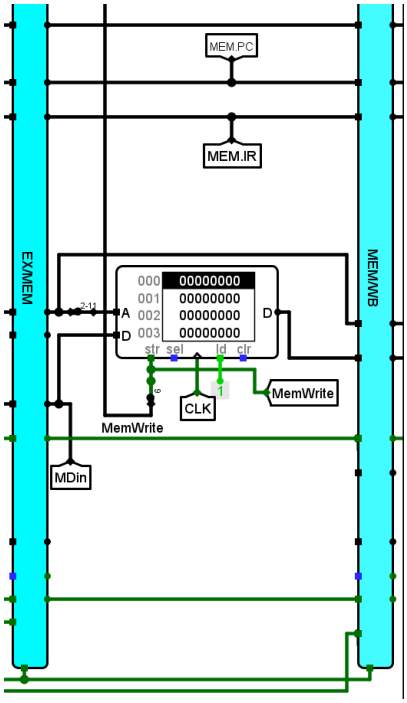
EX.JR 信号控制的多路选择器主要实现 JR 指令，当给出 JR 指令，会将 R[rs]输送到下面的多路选择器，同时给出 JMP=1，从而实现将 R[rs]输入到 PC





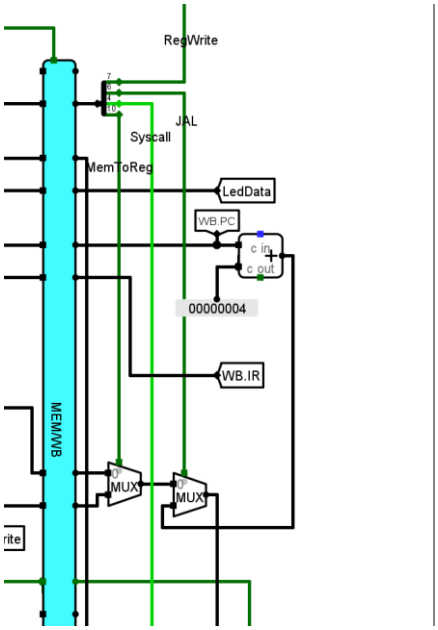
后面两个多路选择器为了实现 beq/bne 指令和 j 指令，利用多路选择器来选择正确的下一条指令地址，这里的 22 常量比较特殊，结合代码可以知道用来标志程序结束，来中止对应的计数部分。

第四阶段：访存 MEM

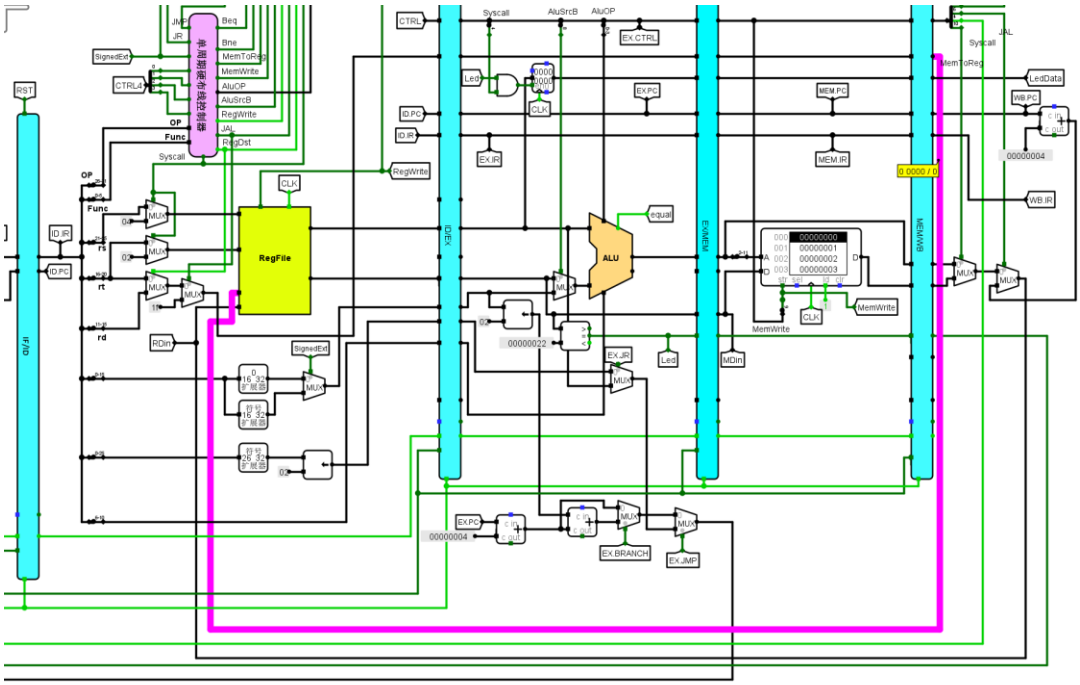


访存阶段根据流水寄存器所提供的地址和控制器所提供的控制信号进行数据读取或进行数据写入

第五阶段：写回 WB



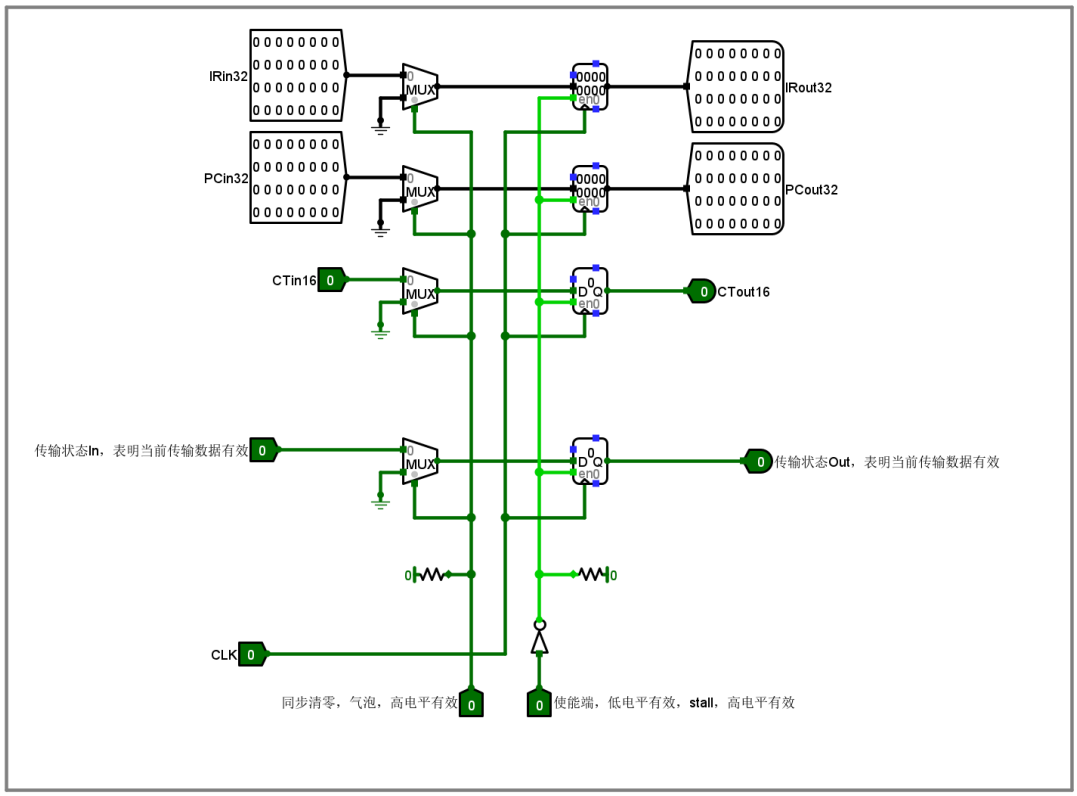
第一个 MUX 的控制信号为 MemToReg，用于选择是将运算器的运算结果还是将内存中的数据写回，第二个 MUX 选择的是将 PC+4 还是将刚刚的选择结果写回，受 Jal 控制信号控制，主要用于实现子程序跳转，将程序断点地址写入到寄存器堆中



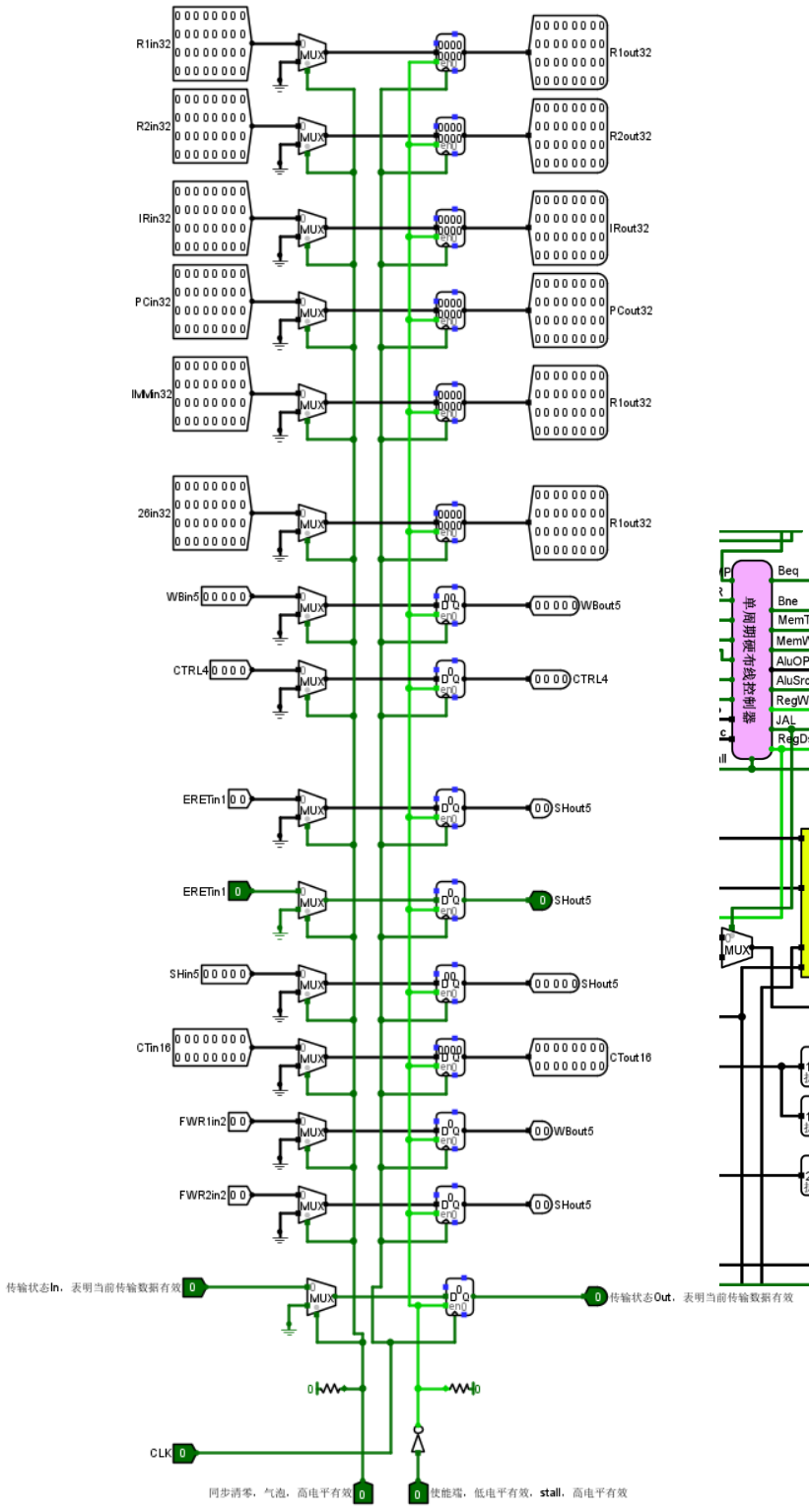
这里为了使写回的寄存器编号可以正确不受后续指令译码结果的影响，会将写回阶段的指令在译码后跟着流水一起流动，直到写回阶段之后修改寄存器堆写入端口，这些步骤在 MEM/WB 寄存器锁存新值再加上电路延迟后完成，但寄存器堆写入需要时钟信号，所以直到下一个时钟信号给出才会完成将数据写入到寄存器中

3.1.3 分析理想流水线 MIPS 处理器的 4 个流水寄存器电路（IF/ID、ID/EX、EX/MEM、MEM/WB）

1.IF/ID 流水寄存器

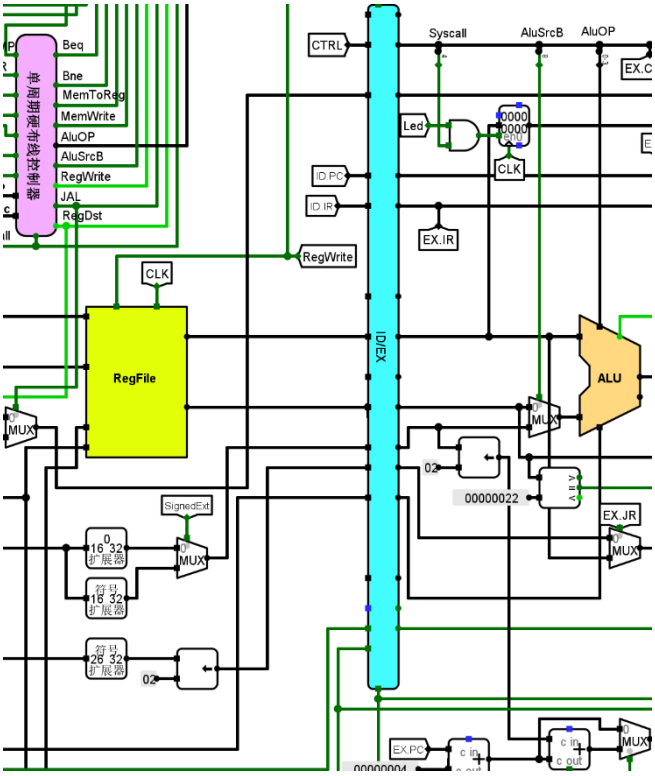


第一个流水寄存器锁存指令和指令的地址，多路选择器的另一端接地，当给出清零信号时会选择输出 0 来清空跳转指令误取的指令，具体放在后续气泡流水线进行详细说明，stall 引脚在程序结束后通过外界信号使流水寄存器中的寄存器停止工作

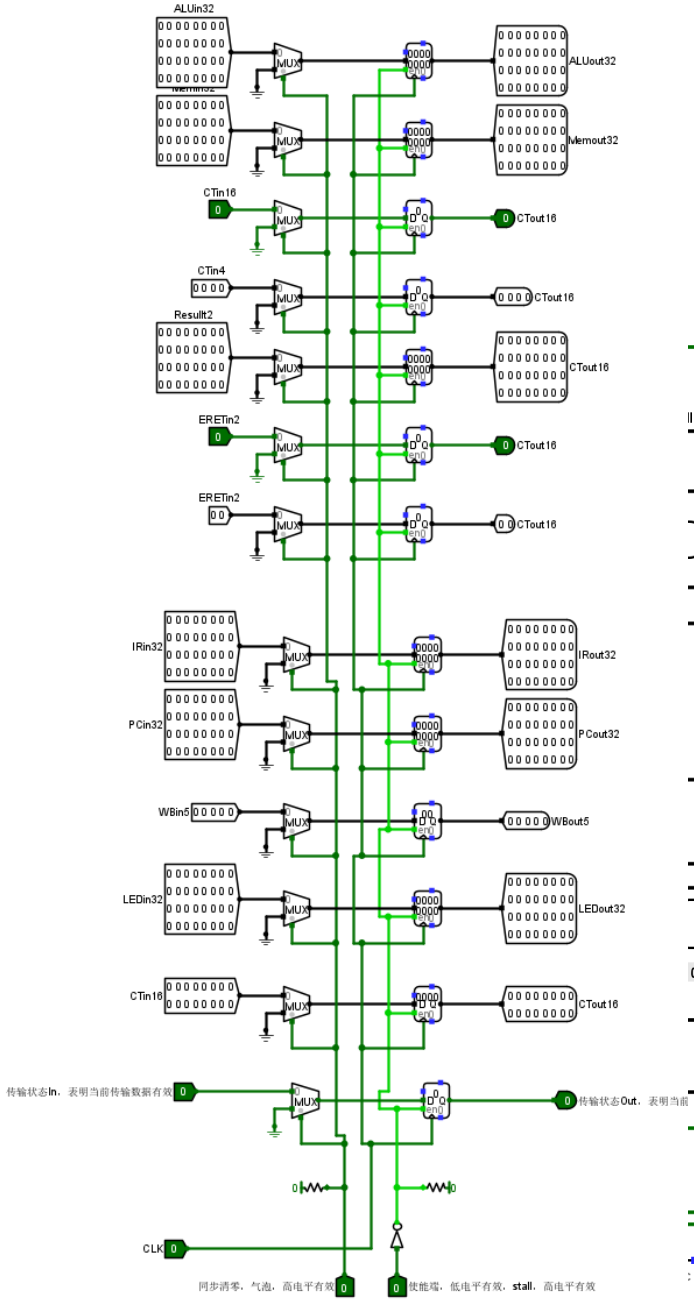


2.ID/EX 流水寄存器

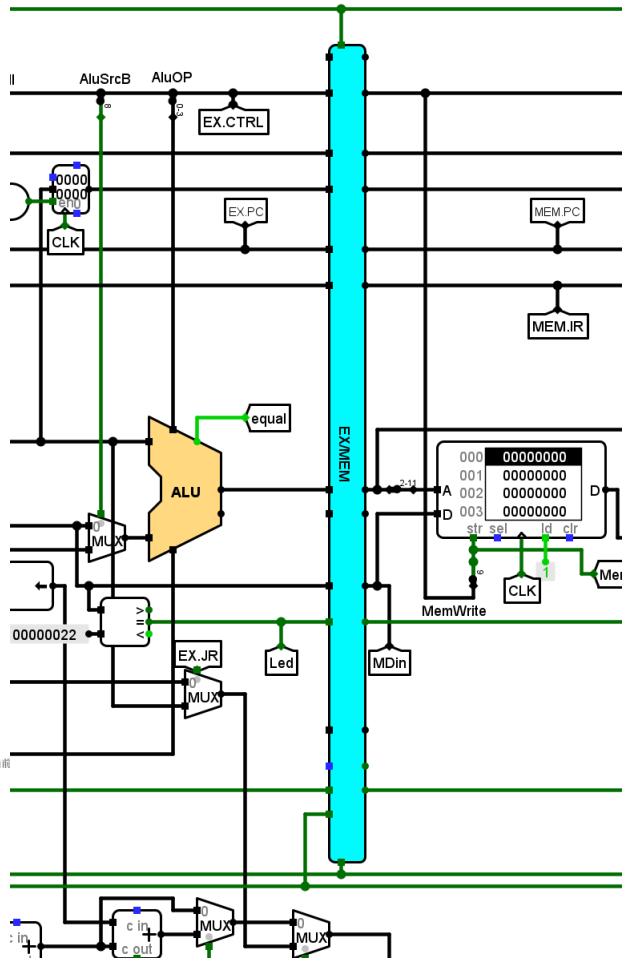
该流水寄存器锁存指令、指令地址、控制信号、译码阶段获取的寄存器中的数据和三种立即数、以及需要在写回阶段需要使用的寄存器（对应在 WBin5 端口），除此之外该流水寄存器和上一个寄存器一样也有一个清零信号来清楚误取的指令和对应的控制寄存器使能的信号，这里的对应寄存器时钟信号也是上跳沿除法，只有当给出上跳沿信号时才会将数据写入到寄存器当中



3.EX/MEM 流水寄存器



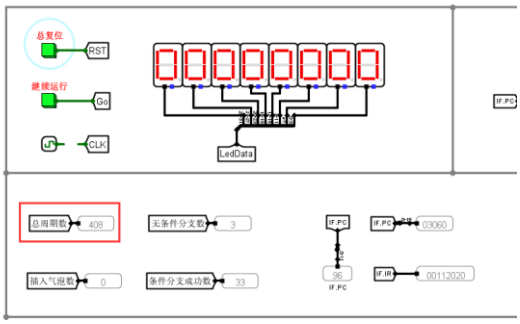
该寄存器锁存从运算器中出来的结果，以及其余访存阶段会用到的数据，该寄存器和前一个寄存器相同也会锁存写回寄存器的编号，用于解决写回时的冲突问题。



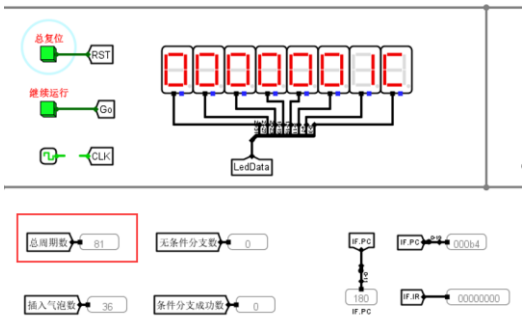
3.2 气泡流水线 MIPS 处理器

3.2.1 在气泡流水线 MIPS 处理器的数据通路上运行 test1.hex、test2.hex、test3.hex、fib_mips.hex、sum_mips.hex、sort1_mips.hex、sort2_mips.hex 等程序，记录每个程序运行后的总周期数

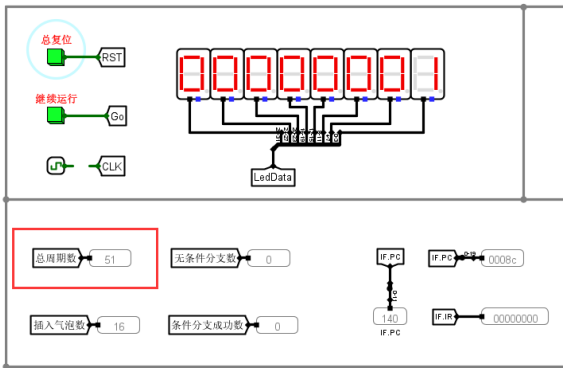
test1.hex：运行时数码管从 1F 减到 0



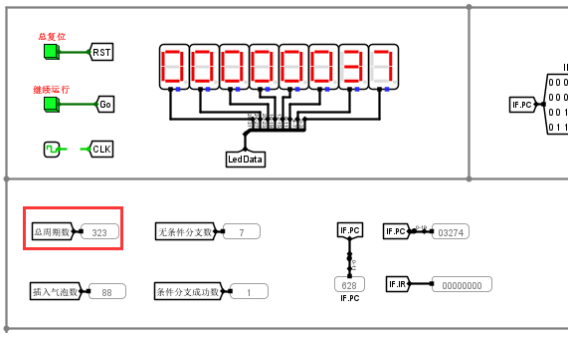
test2.hex：运行结束时数码管显示 0+1+2+…+7=28=1CH



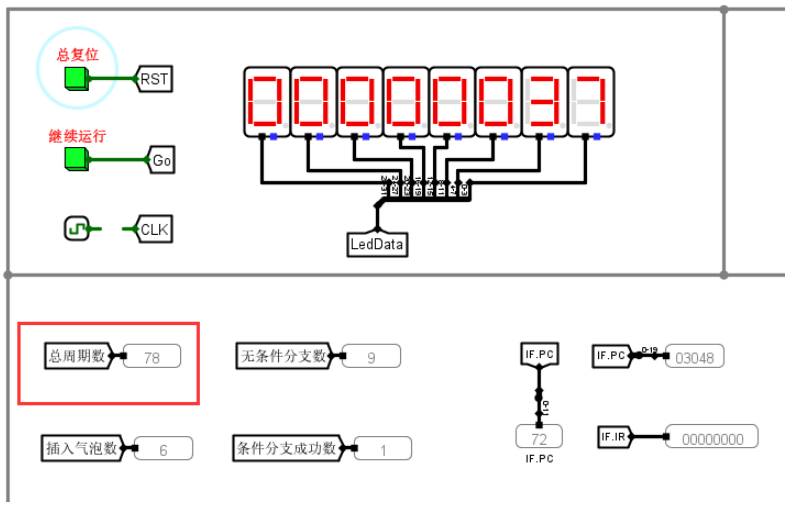
test3.hex：运行时数码管依次显示 1、1、2、5、1



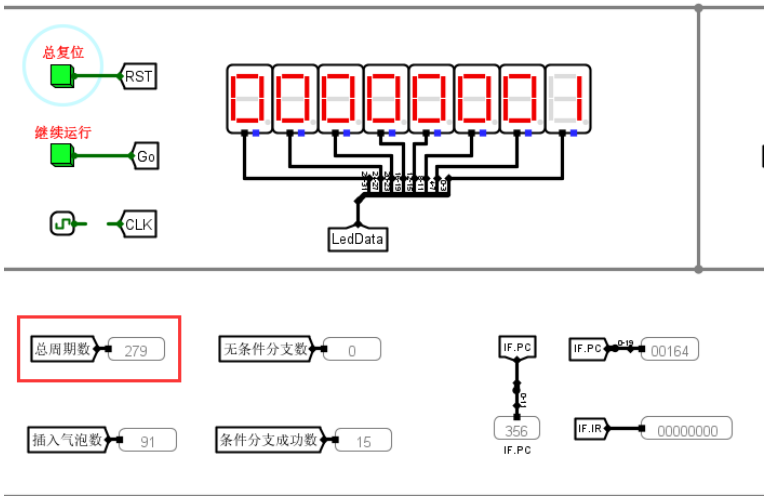
fib_mips.hex：程序运行时会陆续在数码管输出斐波那契数列，同时会在数据存储器中保存对应的数据结果



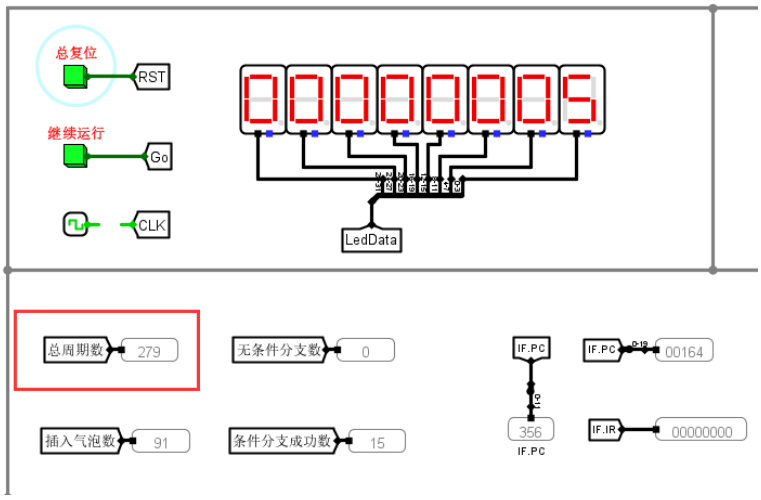
sum_mips.hex：程序运行结束后在数码管输出 37H，即 1+2+3+…+10=55=37H



sort1_mips.hex：在数码管上依次降序输出排序的结果，即依次输出 5、4、3、2、1

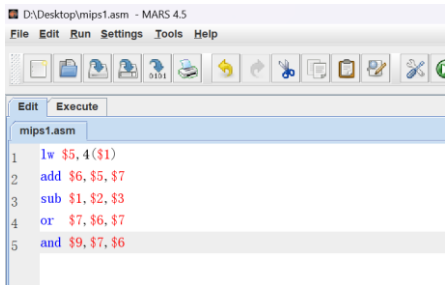


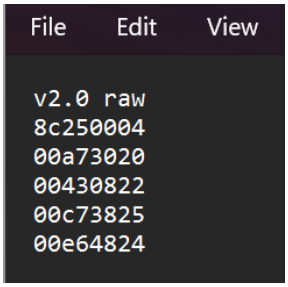
sort2_mips.hex：在数码管上依次升序输出排序的结果，即依次输出 1、2、3、4、5



3.2.2 在气泡流水线 MIPS 处理器的数据通路上运行教材 P269 例 7.1 的程序，给出该程序运行后的时空图，并与教材上的图 7.20 进行比较，观测是否一致？

通过汇编模拟仿真软件生成对应的机器码





在电路中运行可以发现执行完这五条指令按照 add 指令完成数据写回计算使用了 14 个时钟周期（如果只考虑到 add 完成译码阶段则和课本一样为 11 个周期），其对应的时空图如下图所示

CLKs	IF	ID	EX	MEM	WB
1	lw				
2	add	lw			
3	sub	add	lw		
4	sub	add		lw	
5	sub	add			lw
6	or	sub	add		
7	and	or	sub	add	
8	and	or		sub	add
9		and	or		sub
10		and		or	
11		and			or
12			and		
13				and	
14					and

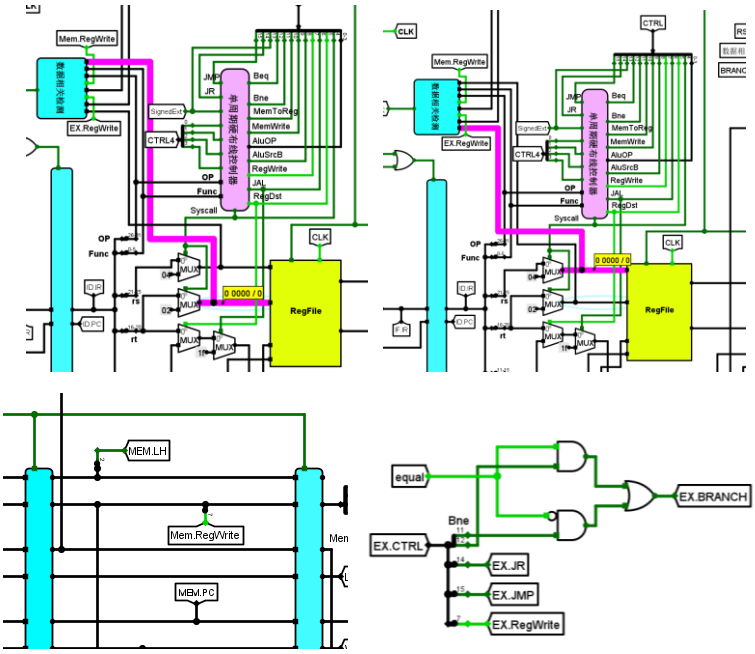
3.2.3 分析气泡流水线 MIPS 处理器的数据通路

数据通路和理想流水线大致相同，这里只分析数据相关检测、控制冲突和数码管数据通路

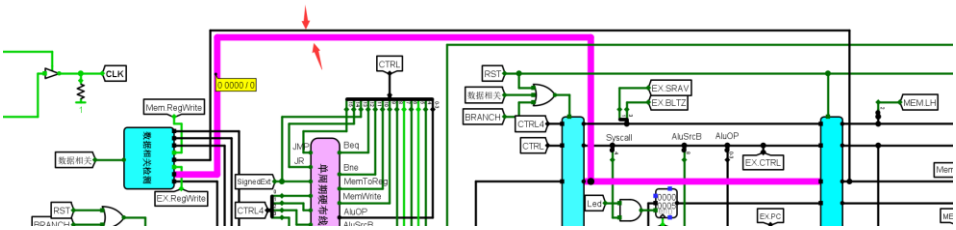
数据相关检测：数据相关性检测需要检测在 ID 阶段指令使用的源寄存器是否为在 EX、MEM 阶段的两条指令写回的寄存器，其中不用考虑 0 号寄存器，因为其值恒为 0 不会产生数据相关性，所以在数据通路中要实现将这些信息传输到对应的数据相关检测器中

```
DataHazzard = RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#)
            + RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#)
            + RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#)
            + RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)
```

该数据通路获取了 ID 阶段的 rs 和 rt 寄存器的编号，同时这里使用了隧道获取了 EX 和 Mem 阶段是否会发生写回

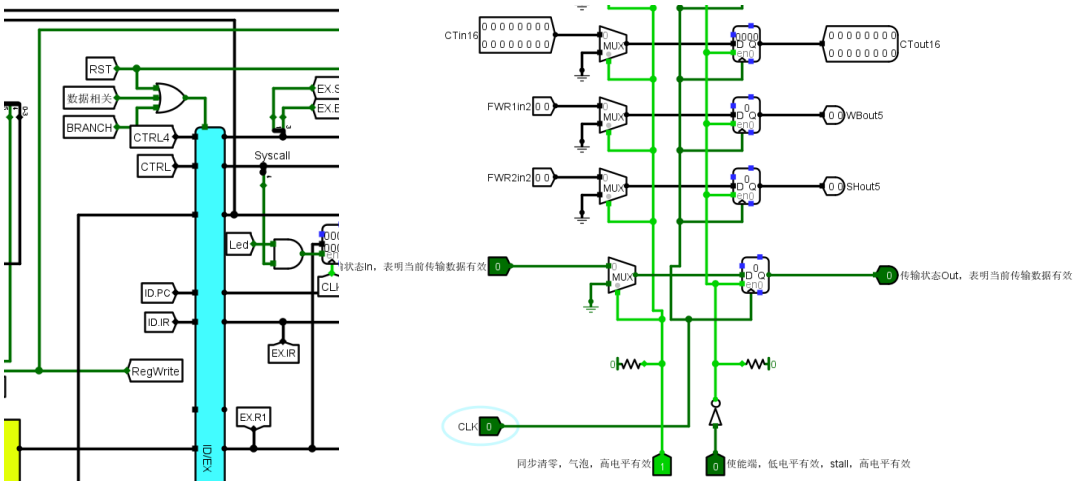


最后通过数据通路获取 EX 和 MEM 阶段写回的寄存器编号传入数据相关检测器当中，数据相关性检测具体实现放到后面分析

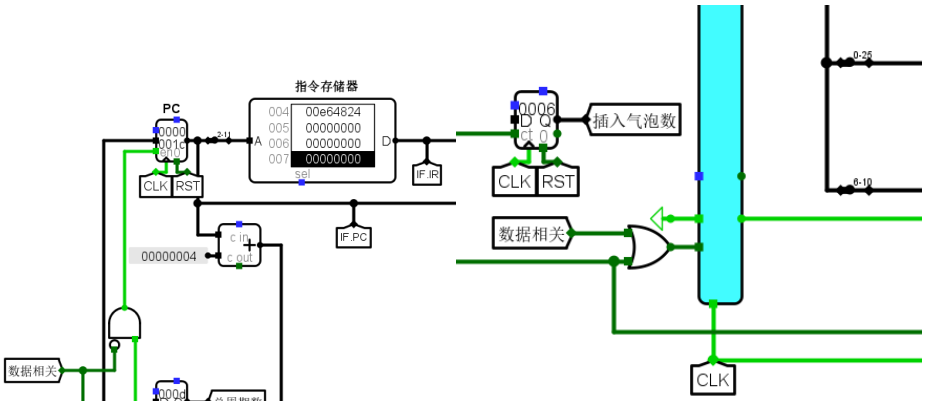


如果发生数据冲突就需要往流水线中插入两个气泡，插入两个气泡只需在下面的两个时钟周期内保证 ID/EX 寄存器中是空值，并且这两个时钟周期内 PC 不会自加即可

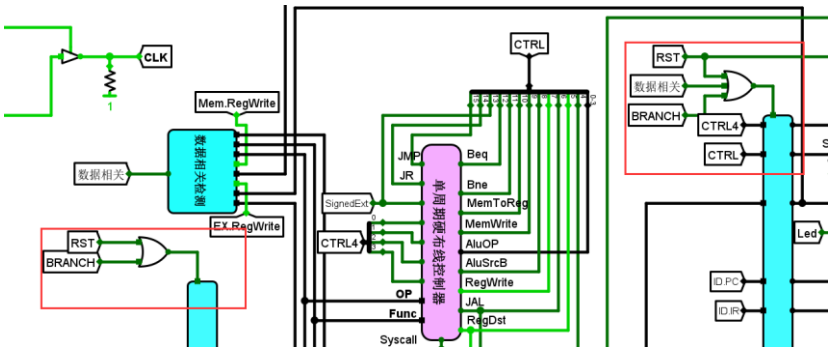
发生数据冲突会向 ID/EX 流水寄存器发送一个 Flush 信号，在该信号下所有输入端口的多路选择器都会输出接地电平即 0，完成指令的清除，第二个时钟周期时该气泡会在流水线下流向下一个阶段



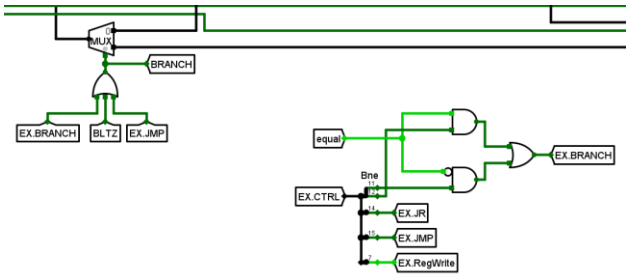
将数据相关的检测结果直接取反输入 PC 的使能端即可完成 PC 的暂停，当两个时钟周期后 ID/EX 和 EX/MEM 流水寄存器中值为空值时数据相关检测为，PC 继续开始自加操作,同时这里也会将 IF/ID 流水寄存器的使能端置为 0 来使该寄存器暂停工作。



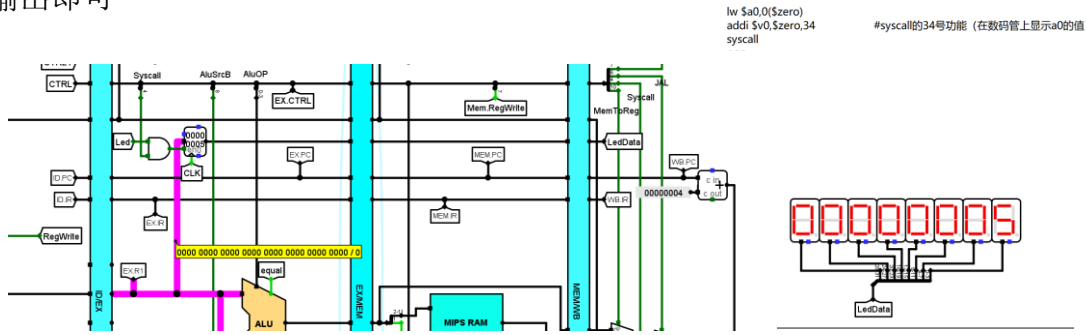
控制冲突：控制冲突的处理和数据冲突的相同，只需要清空误取的两条指令，即将 IF/ID 和 ID/EX 流水寄存器中的内容清空，这里通过 branch 来传输跳转清空信号



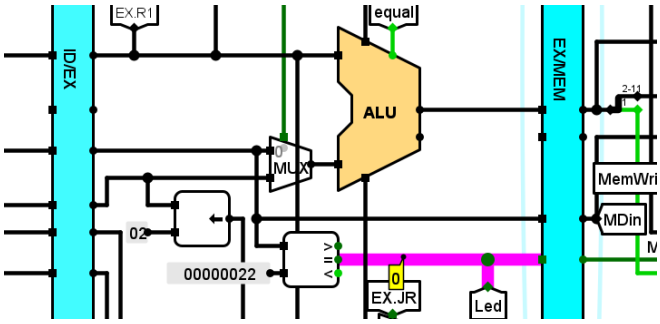
通过比较各部分的跳转信息生成是否跳转信号 branch



数码管：数码管的数据通路比较简单，直接获取 rs 对应的数据之后通过流水线输出即可



判断系统调用号是否为 34（22H），从而判断是否要修改数码管的值

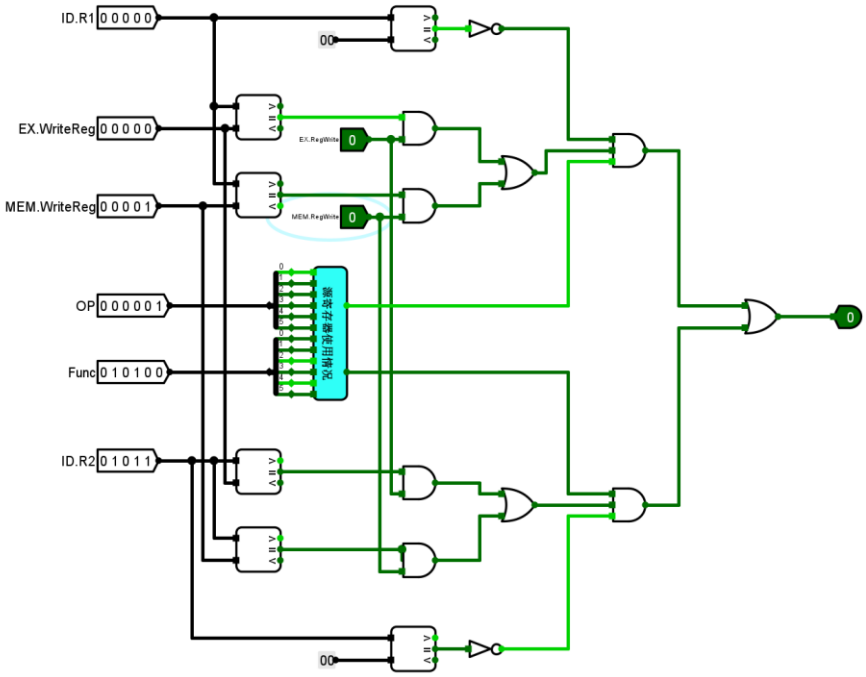


3.2.4 分析气泡流水线 MIPS 处理器的数据相关检测电路

该部分要实现的逻辑表达式如下图所示

```
DataHazzard = RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#)
            + RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#)
            + RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#)
            + RtUsed & (rt≠0) & MEM.RegWrite & (rt==MEM.WriteReg#)
```

因此对输入的结果通过比较器和逻辑门即可实现对应的效果，先判断写回的寄存器是否为 0 号寄存器，之后判断对应阶段是否会发生数据写回，如果是，先通过源寄存器使用情况判断 rs 和 rt 寄存器是否被使用，如果使用的话就和对应的写回寄存器编号进行与运算即可，最终生成数据相关性检测结果，这里的源寄存器使用情况直接根据组合逻辑通过 OP 和 Func 生成结果即可

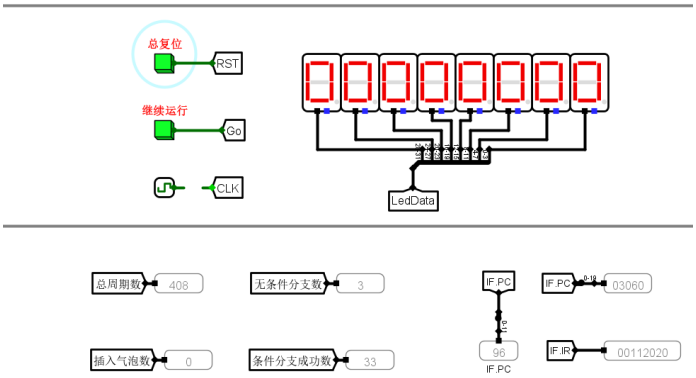


3.3 重定向流水线

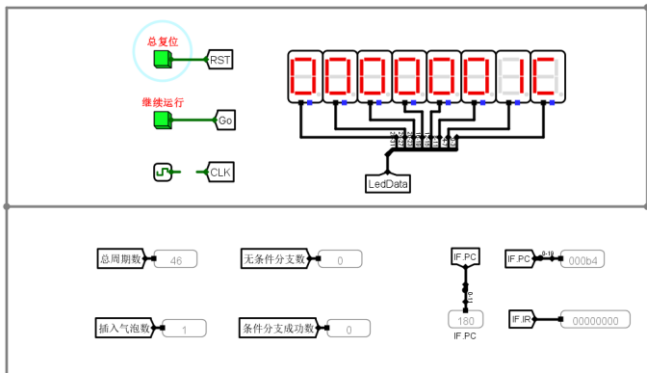
3.3.1 在重定向流水线 MIPS 处理器的数据通路上运行 test1.hex、test2.hex、test3.hex、fib_mips.hex、sum_mips.hex、sort1_mips.hex、sort2_mips.hex 等程序，记录每个程序运行后的总周期数，并与气泡流水线对应程序的总周期数进行比较，得出什么结论？

测试程序的运行效果和气泡流水线的相同，下面只比较重定向流水线的时钟周期数

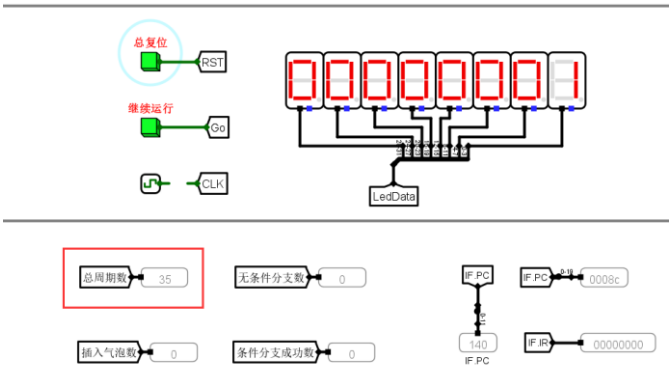
test1.hex：在气泡流水线中总周期数为 408，该段程序没有发生因为数据相关性插入气泡，因此在两个程序中的总周期数相同



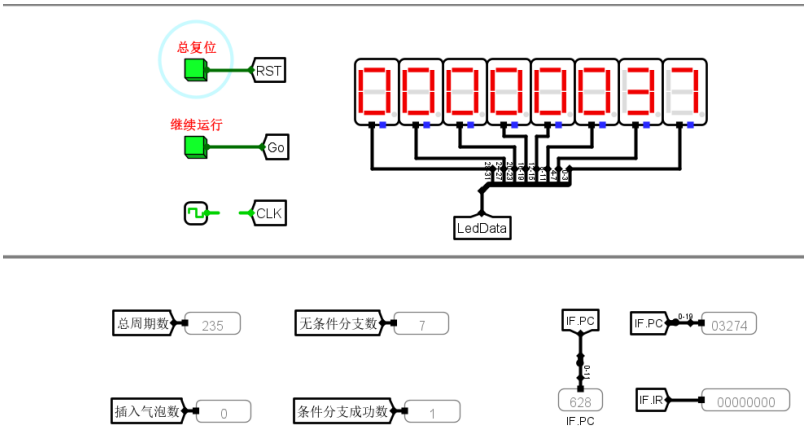
test2.hex：在气泡流水线中总周期数为 81，插入了 36 个气泡，该段代码有一个 Load-USE 指令，因此在重定向流水线中会插入一个气泡，所以在重定向流水线中时钟周期数位 46



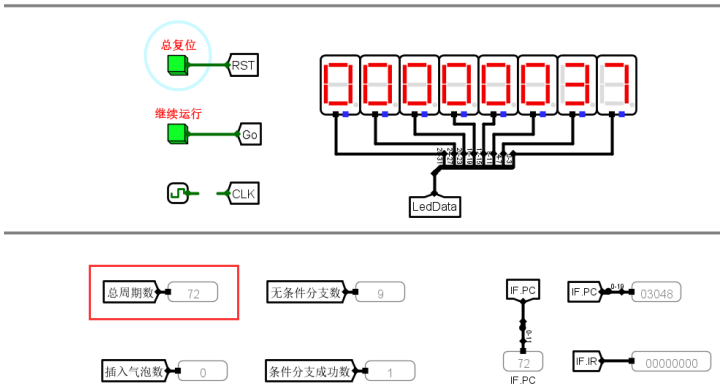
test3.hex：在气泡流水线中总周期数为 51，插入了 16 个气泡，重定向流水线没有插入气泡，所以总周期数为 35



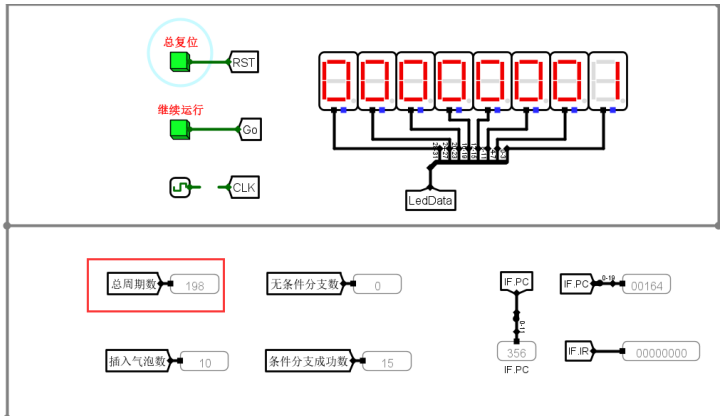
fib_mips.hex：在气泡流水线中总周期数为 323，插入了 88 个气泡，在重定向流水线中没有插入气泡



sum_mips.hex：在气泡流水线中总周期数为 78，插入了 6 个气泡，重定向流水线中没有插入气泡



sort1_mips.hex：在气泡流水线中总周期数为 279，插入了 91 个气泡，这里出现了 10 次 Load-Use 指令，因此要插入 10 个气泡



sort2_mips.hex：同上

由此可见，通过重定向流水线的方式消除了大部分因为数据相关性产生的气泡，大大提高了流水线的效率。

3.3.2 在重定向流水线 MIPS 处理器的数据通路上运行教材 P275 例 7.2 的程序，给出该程序运行后的时空图，并与教材上的图 7.28 进行比较，观测是否一致？

程序运行中插入了一个气泡，对应的时空图如下图所示

CLKs	IF	ID	EX	MEM	WB
1	lw				
2	add	lw			
3	sub	add	lw		
4	sub	add		lw	
5	or	sub	add		lw
6	and	or	sub	add	
7		and	or	sub	
8			and	or	sub
9				and	or
10					and

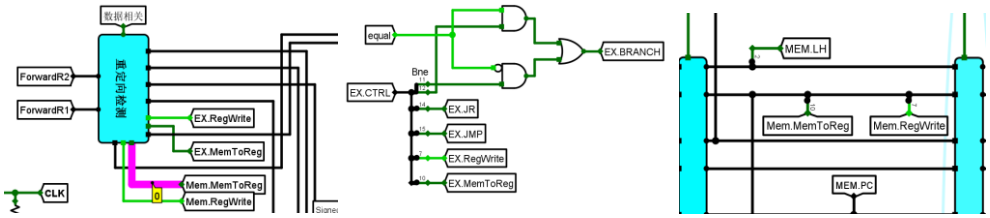
3.3.3 分析重定向流水线 MIPS 处理器的数据通路

分为两部分，一部分是生成控制信号的数据通路，另一部分是重定向数据通路

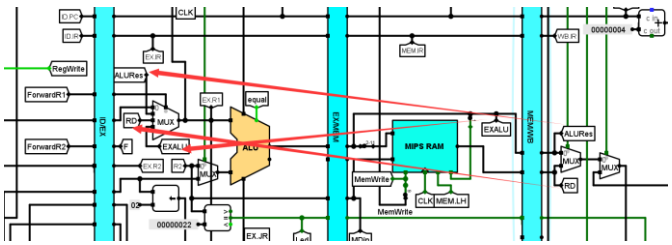
控制生成信号：三个重定向有关的检测逻辑如下图所示，需要在数据通路中将这些数据传输到检测逻辑当中，其中前两个相关输入的数据通路和气泡流水线完全相同

```
if (RtUsed & (rt≠0) & EX.RegWrite & (rt==EX.WriteReg#) )   RtForware=2
else if ( RtUsed & (rt≠0) & MEM. RegWrite & (rt==MEM.WriteReg#)   RtForware=1
else   RtForware=0

if (RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#) )   RsForware=2
else if ( RsUsed & (rs≠0) & MEM. RegWrite & (rs==MEM.WriteReg#)   RsForware=1
else   RsForware=0

LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#)
        + RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)
```

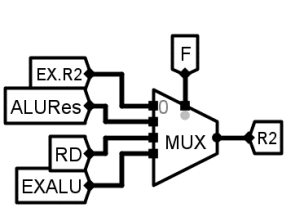
EX、MEM 阶段读取内存的控制信号直接在对应阶段获取，之后传输到重定向检测当中

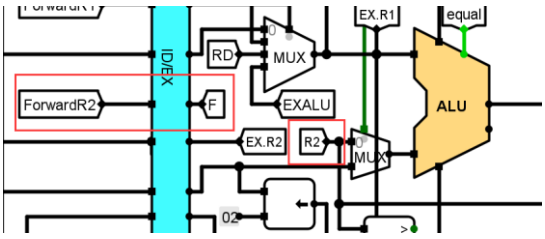


重定向：运算器第一个操作数的重定向数据通路如图所示，多路选择器的输入端依次为 ID 阶段的输出、写回阶段的 ALURes、写回阶段的 RD 和执行阶段运算器的计算结果 EXALU，之后选择信号由 ForwardR1 给出，具体映射方式在后面说明

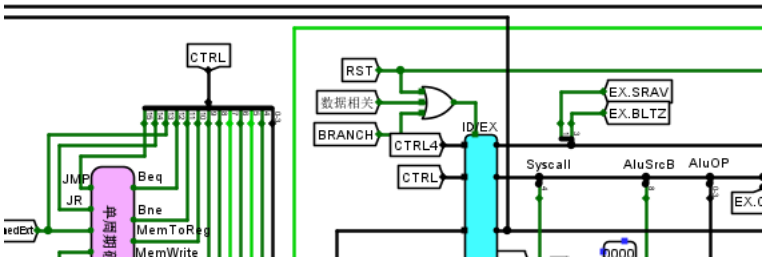


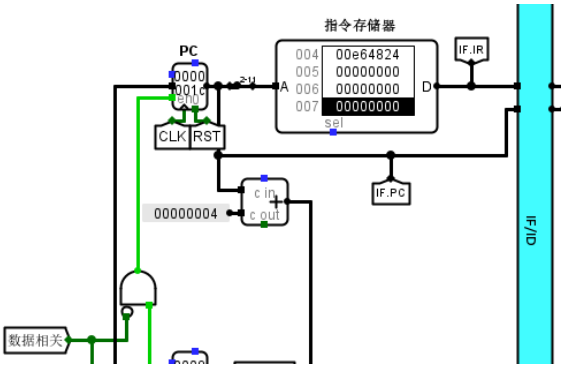
第二个操作数的选择和第一个操作数原理相同，这里由于电路过于复杂因此将对应的部件通过隧道移到了外边，但是原理基本相同





气泡插入：只有当出现 Load-Use 指令会插入一个气泡，此时数据相关控制信号被置为 1，接下来的处理情况和气泡流水线完全相同，也是通过控制 ID/EX 的所有输入为 0 从而实现清空该流水寄存器，并且同时会使 PC 的使能端设为 0 来使 PC 停止工作不再读入下一个地址



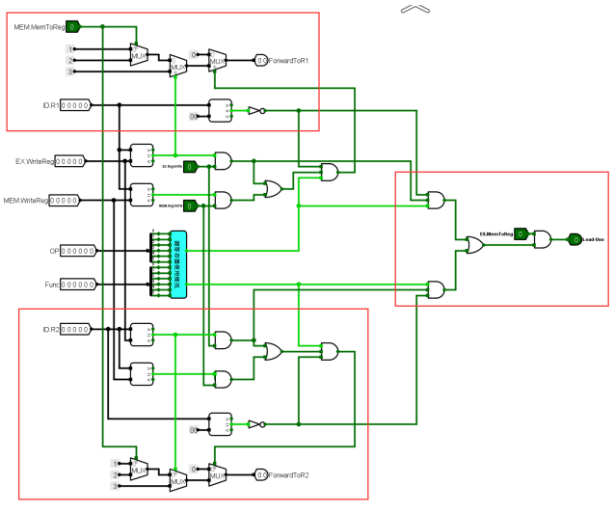


3.3.4 分析重定向流水线 MIPS 处理器的重定向检测电路

重定向控制信号会在译码阶段结合前面的两条指令直接生成并随着流水线进入和指令一起进入执行阶段，重定向检测电路可以分为两个部分，其中一个部分用于生成运算器输入端多路选择器的选择信号，另一部分生成 Load-Use 指令的判断结果。

这里三个选择器从右到左看，最右边的多路选择器会根据是否存在数据相关性选择 0 或其他值，对应选择信号来源和气泡流水线的相同，比较 EX、MEM 阶段是否写回并且查看如果写回的话写回寄存器的编号和译码阶段的编号是否相同，如果不发生冲突就输出 ForwardToR1=0，代表没有数据相关直接将前一个流水寄存器中的数据传输。

如果存在数据相关将从 1、2、3 中选择；第二个多路选择器将用来判断是和 MEM 阶段还是和 EX 阶段指令发生冲突，这里只需要将译码阶段的端口和 EX.WriteReg 比较即可，因为已经保证了这一轮选择的前提是已经存在数据相关，两个比较器输出同时为 1 时会优先输出 3，实现优先使用 EX 阶段 ALU 生成的最新结果；最后一个多路选择器选择的是来自 MEM 重定向的数据是从内存出来的还是直接由运算器运算得到的



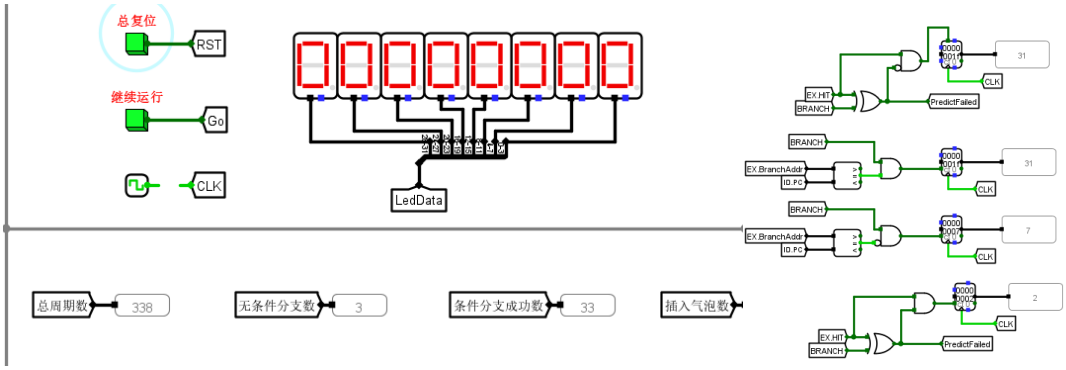
第二个红框内的电路负责生成 Load-Use 控制信号，根据下面的逻辑表达式结合分配律，只需要将两个数据相关性检测结果作或运算再和 EX.MemRead 做与运算即可

$$\begin{aligned}\text{LoadUse} = {} & \text{RsUsed} \,\&\, (rs\neq 0) \,\&\, \text{EX.MemRead} \,\&\, (rs==\text{EX.WriteReg\#}) \\ & + \text{RtUsed} \,\&\, (rt\neq 0) \,\&\, \text{EX.MemRead} \,\&\, (rt==\text{EX.WriteReg\#})\end{aligned}$$

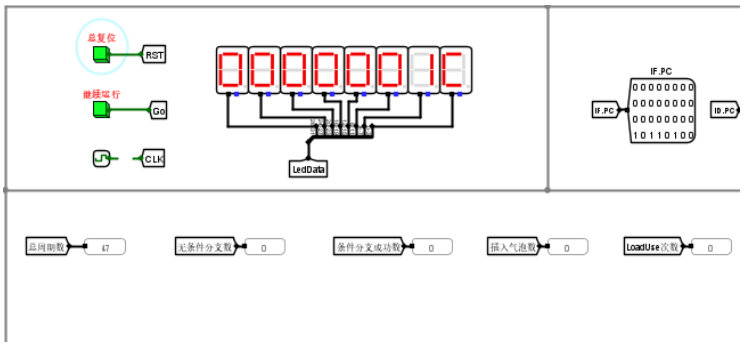
3.4 动态分支预测流水线

3.4.1 在动态分支预测流水线 MIPS 处理器的数据通路上运行 test1.hex、test2.hex、test3.hex、test4.hex、fib_mips.hex、sum_mips.hex、sort1_mips.hex、sort2_mips.hex 等程序，记录每个程序运行后的总周期数，并与气泡流水线、重定向流水线对应程序的总周期数进行比较，得出什么结论？

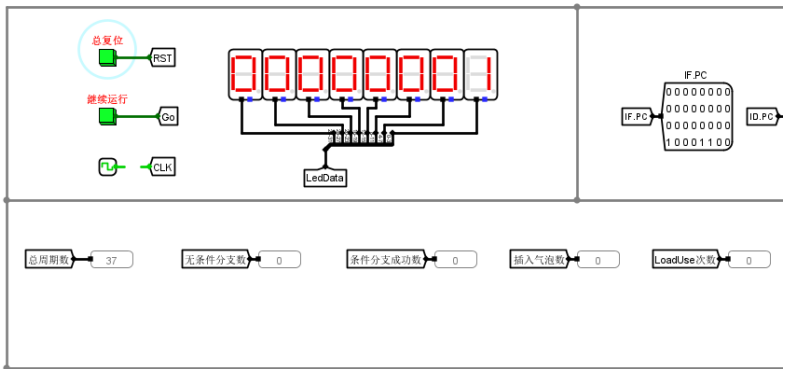
test1.hex：



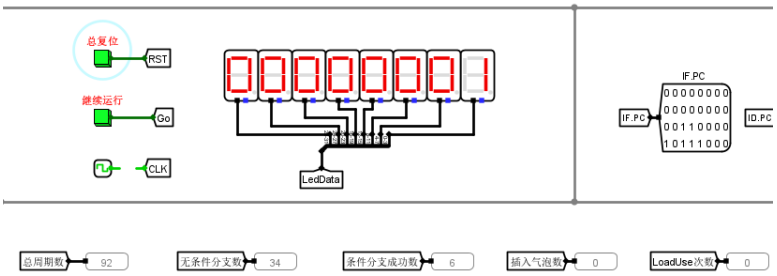
test2.hex：



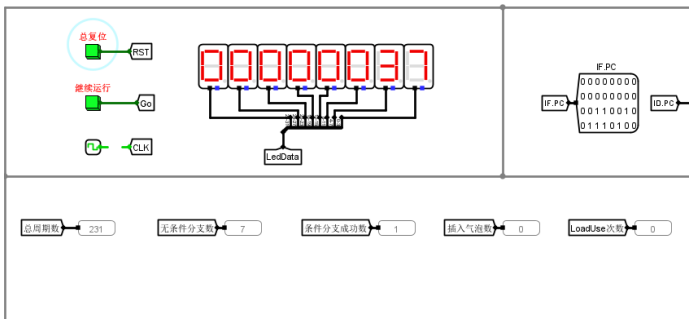
test3.hex：



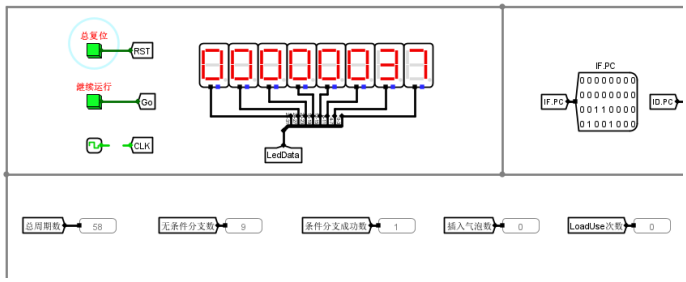
test4.hex：



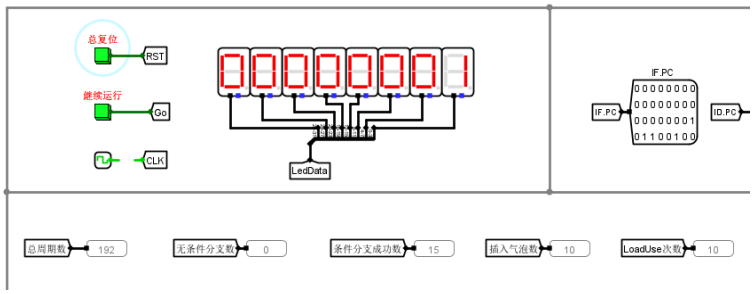
fib_mips.hex



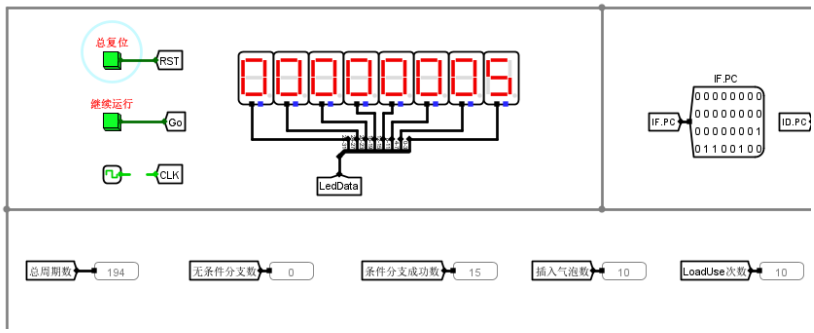
sum_mips.hex



sort1_mips.hex



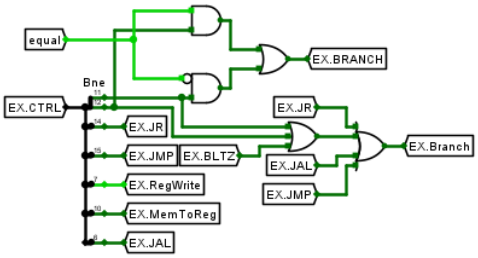
sort2_mips.hex



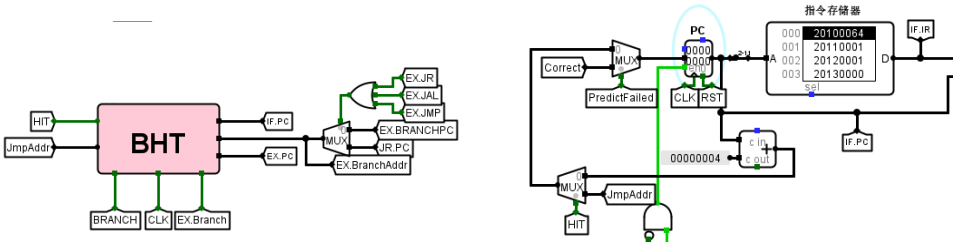
通过动态分支预测之后，流水线中的控制冲突会大大减少，尤其对于有多次循环的程序（如斐波那契数列和求和等），根据预测结果使得可以减少发生清空误取指令的机会，大大提高了流水线的效率。

3.4.2 分析动态分支预测流水线 MIPS 处理器的数据通路

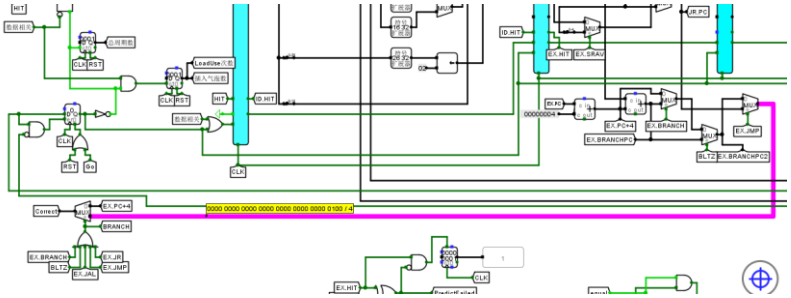
该部分为执行阶段的有条件跳转是否跳转信号生成部分，通过运算器的状态输出来实现条件判断，所生成的跳转结果为 EX.BRANCH，后续会用于 BHT 表的更新。



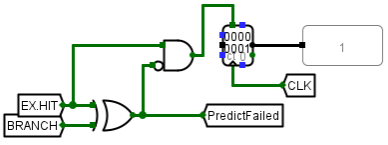
接下来从 PC 开始分析动态预测技术的实现，当通过 PC 获取一条指令的同时，该指令的地址会同步进入 BHT 中查找其对应的预测结果，HIT 表示是否发生跳转，HIT=1 说明预测跳转，HIT=0 预测不跳转，通过 Hit 作为选择信号来选择 PC+4 和预测的跳转地址，当下一个时钟周期到来时 PC 就会根据 HIT 选择的地址来执行。这里还有另一个多路选择器，其选择端为 PredictFailed，用于根据预测结果是否正确来修正 PC 的值，后续进行说明。



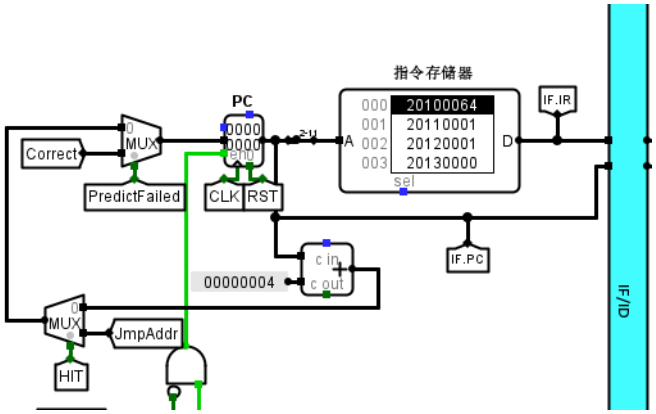
之后 HIT 会和指令一起在流水线上进入 EX 阶段，该阶段会生成该条指令的实际跳转情况，如果发生了跳转那么就会将跳转的地址存入 Correct，否则会将 PC+4 存入 Correct，代表实际应该选择的地址，这里将所有跳转合并成一个 BRANCH，只要 BRANCH=1 就代表实际是需要跳转的，该跳转可能包括有条件、无条件、子程序等多种类型中的一个。



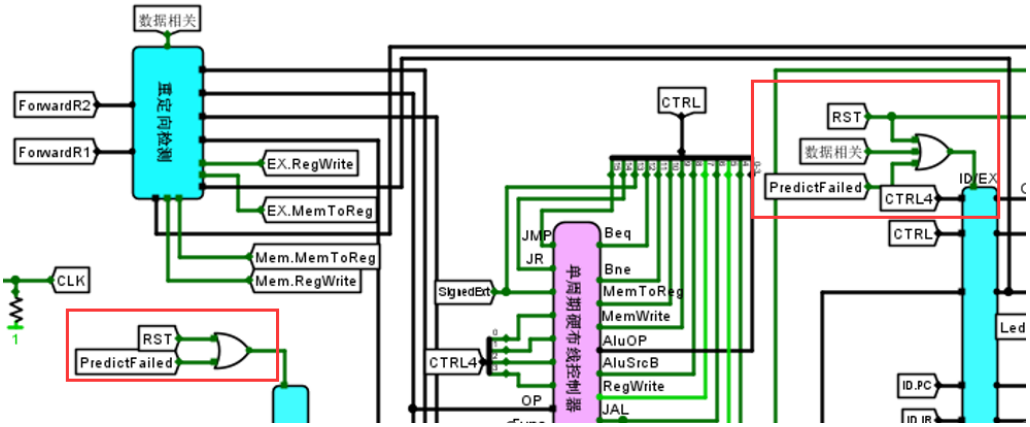
之后得到预测结果，只需要将预测跳转结果 HIT 和实际跳转结果进行异或即可，如果 HIT=0 但是跳转了说明预测失败，或者 HIT=1 但是没有跳转也是预测失败，这个时候需要重新修改 PC 的值来实现程序的重新进行



接下来的电路是根据预测结果来修正 PC，当预测失败时需要将指令纠正，就是将实际运行的指令地址 Correct 重新装入 PC 中；如果预测是正确的那么直接按照接下来指令及预测结果进行即可。

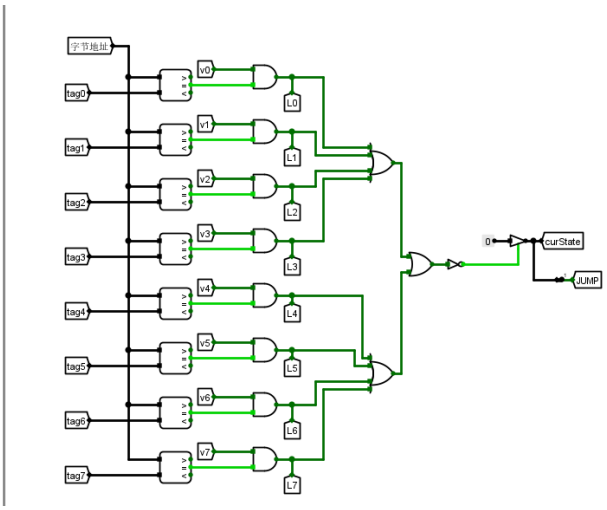


而一旦预测失败除了要调整 PC 之外还需要清除流水线中误取的指令，和前面的气泡流水线相同使用 Flush 信号将对应的流水寄存器清 0 即可完成误取指令的清除，这个时候流水线中会出现两个气泡。

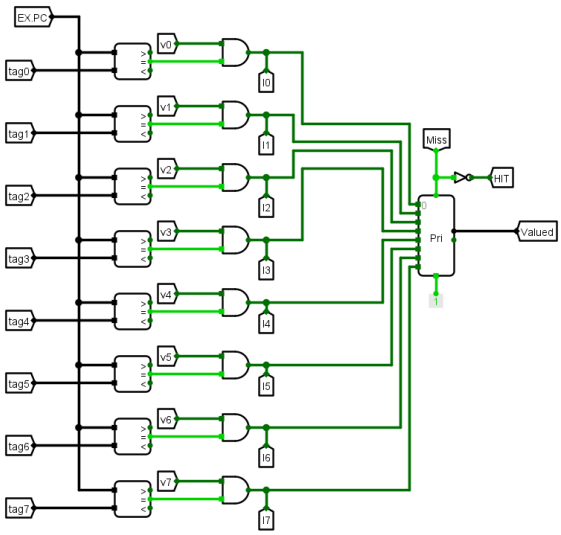


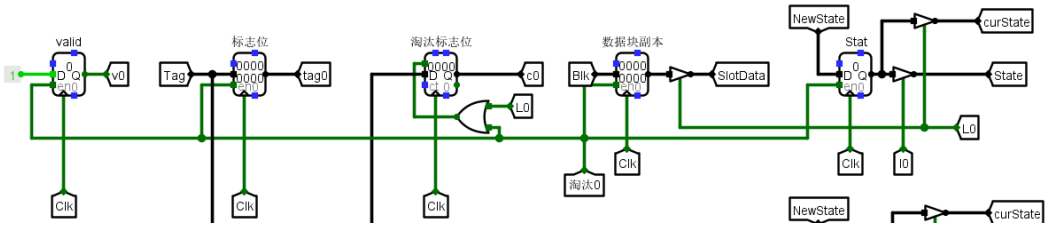
3.4.3 分析动态分支预测流水线 MIPS 处理器的 BHT（分支历史表）电路

指令的查找：先根据 8 路并发比较的方式进行查找，这里的每一个 Tag 就是在 cache 中存储的指令地址，如果查找到对应的结果就会读取对应数据块中的分支地址，之后根据置换标志 state 来输出是否需要跳转，如果 state=11 或 10 则预测跳转，否则预测不跳转



置换标志修正：当执行阶段获取了实际的跳转情况之后，会进行 BHT 表项中的状态修改。首先利用 EX.PC 段的指令进行并发查找，获取对应该条指令在 cache 中的存储位置，之后利用查找到的数据块编号获取该数据库中存储的二位预测状态数据，只需要将对应 state 三态门打开即可

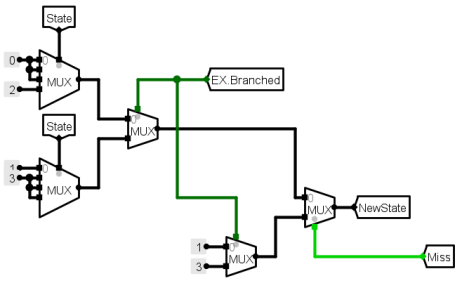


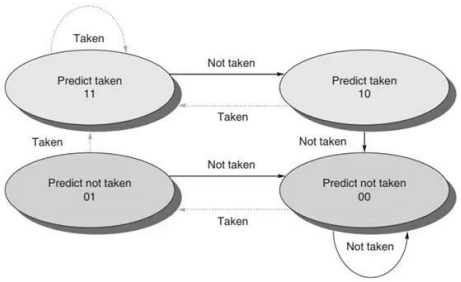


之后对状态进行更新，如果该指令在 cache 中缺失，会先将其写入并设置初始值，如果这条指令是跳转的则设置初始值为 11，否则设置为 01，这个可以根据实际情况改变，但选择合适的初始值可以增加预测的准确率。

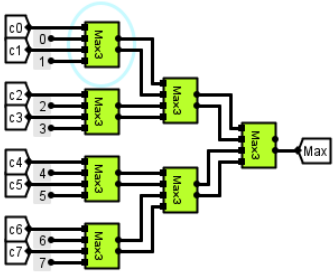
如果该指令在 cache 中是存在的，就要进行状态更改，用表格的形式体现修改方式（原电路这一部分有错误），下图电路为改正之后的

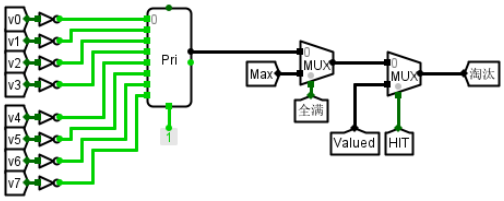
State 初始值	执行阶段命中	修改后的状态
00	0	00
01	0	00
10	0	00
11	0	10
00	1	01
01	1	11
10	1	11
11	1	11



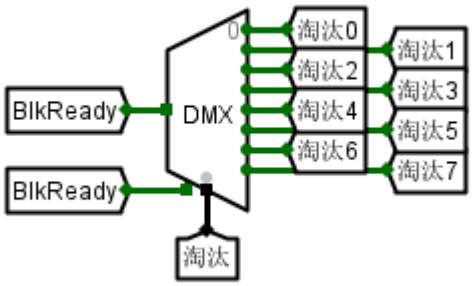


Cache 淘汰算法：使用 **LRU** 算法，当所有 cache 块都已经满了的时候给出全满信号，之后选出计数器值最大的 cache 块进行淘汰

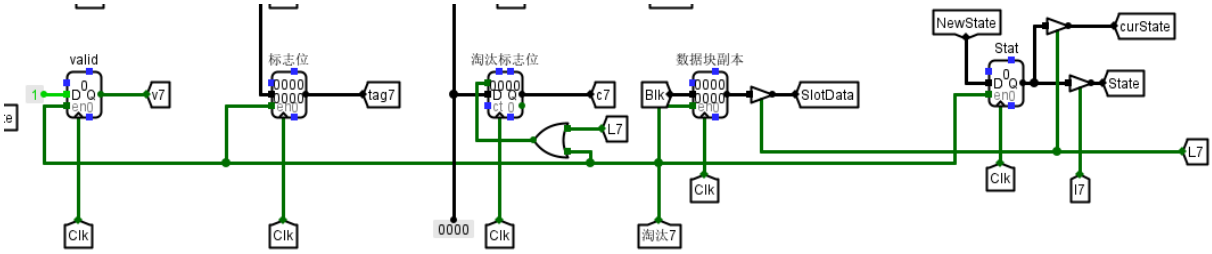




使用解复用器获得淘汰的具体数据块



淘汰信号被置为 1 时对应的 cache 块被选中，完成新数据的写入，完成了 cache 新数据的载入和旧数据淘汰



之后在流水线流动时，BHT 表会对 IF 指令进行是否跳转预测，并且如果跳转输出跳转的地址，同时会根据执行阶段指令的跳转情况进行查找和状态更新，二者同时进行完成动态预测。

外部流水线电路利用判断结果预测是否跳转，大大减少了清空误取指令的情况，从而提高了流水线的效率。