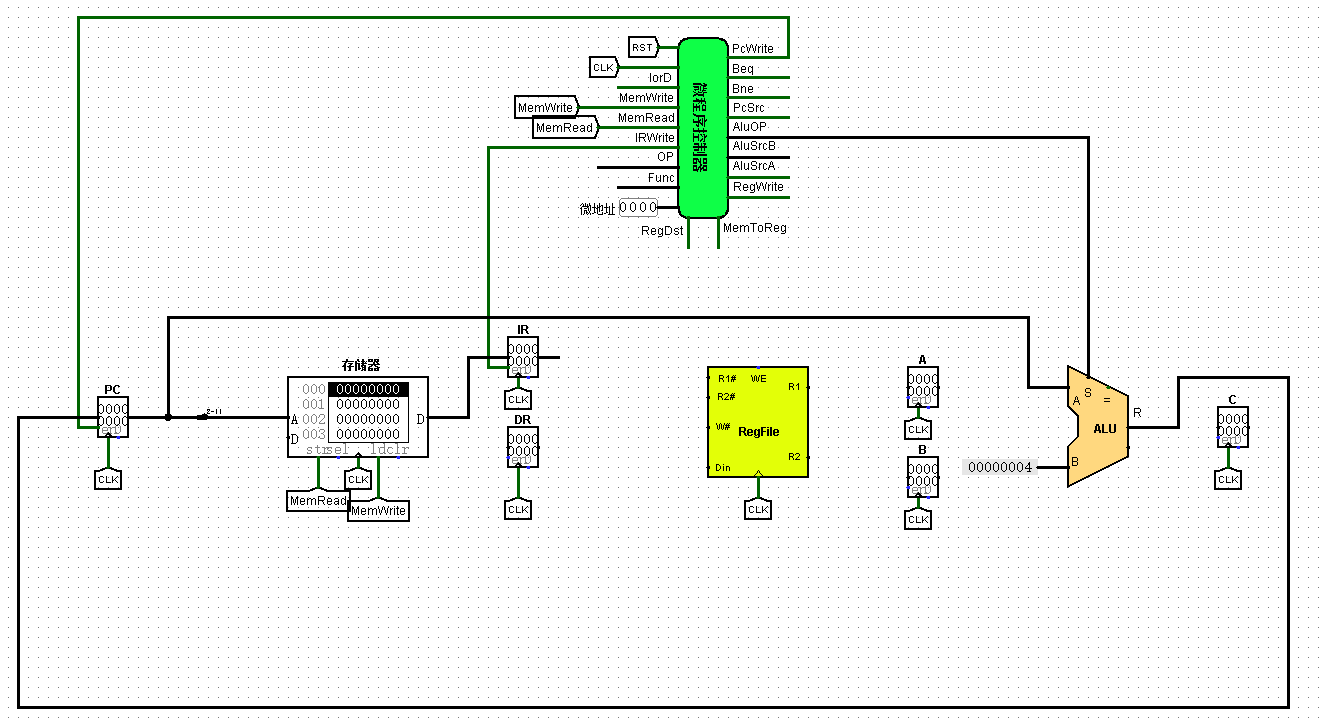
1. 取指周期

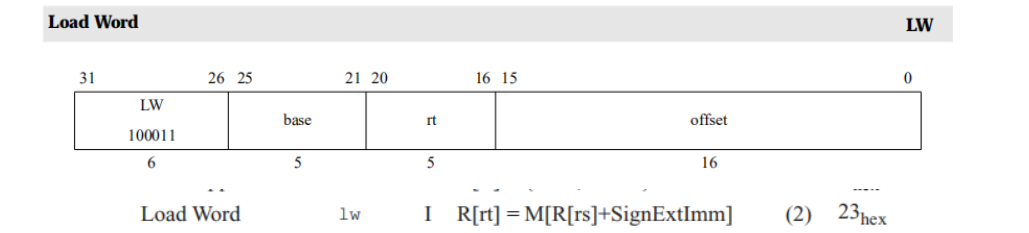
所有指令都一样，第一个节拍到来，PC数据作为地址送入存储器，存储器读信号MemRead产生，存储器通过地址和读信号输出指令数据到IR寄存器，IR寄存器是带有写使能端的（这样才能在其他节拍保存IR持续供给指令信号给控制器和其他各部分），所以写入还需要产生IRWrite信号，

同时因为运算操作比访存操作快得，所以PC+4->PC也进行完毕并且更新PC（开始的时候PC输出但是Mem无读信号，所以没有指令，但是同时PC也输出到ALU+4输出到PC端，这样一旦来了CP信号，和PC写信号PC就会更新，同时由于读信号IR写信号的产生初始指令存储到了IR中），同时我们注意到要是想使用ALU而不再添加加法器，我们还需要一个AluOP信号选择加法操作，



1. 指令分析对IR分位
   1. J指令：实验设计的8条指令无J指令
   2. I指令：LW、SW、BEQ、BNE、ADDI

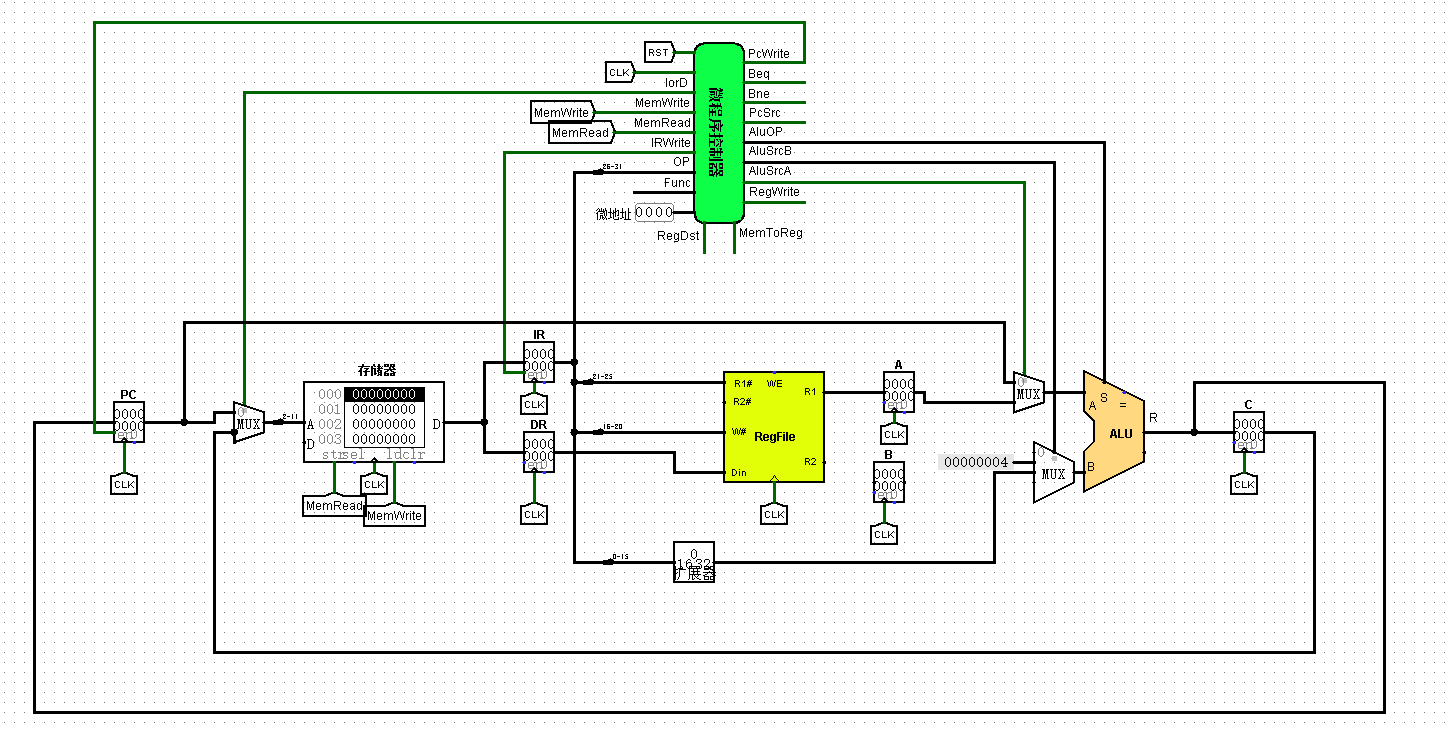


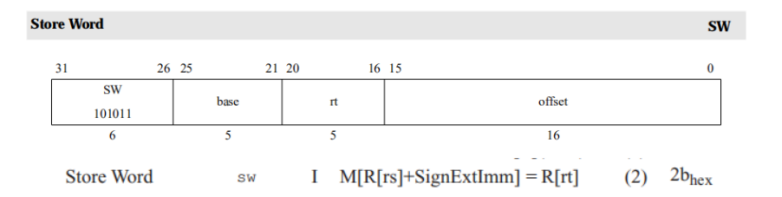
LW：

执行第一个节拍：取出R[rs]存入寄存器A，并且输出到ALU的A端，这时候我们发现同一输入源有多个来源，添加控点AlusrcA，位拓展器的数据输入B端，同理添加控点AlusrcB，

第二个节拍到来数据被保存在寄存器C同时送入存储器地址端，同理增加控点lorD，此时产生MemRead信号，使得数据输出，

第三个节拍到来数据被读入DR。同时输入到Din端，第四个节拍寄存器写信号到来写入寄存器堆

结果：（写信号忘记连了）

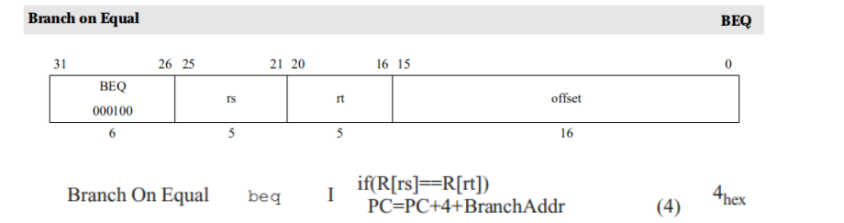
SW：

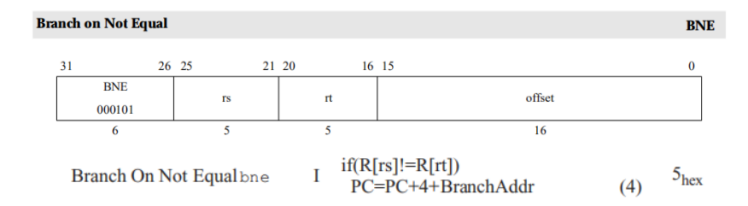
同理分析得到：增加数据输入B寄存器和从B寄存器输入到存储器。

第一个节拍：A寄存器保存数据、B寄存器保存数据，同时A寄存器输入到ALU的A端触发**ALUsrcA**，B端选择扩展数据触发ALUsrcB，产生输出。B寄存器输出到存储器的Din端

第二个节拍到来ALU输出数据保存到C寄存器，并且输入到存储器A地址端需要信号IorD，

第三个节拍到来产生**MemRead信号，写入数据**

BEQ:

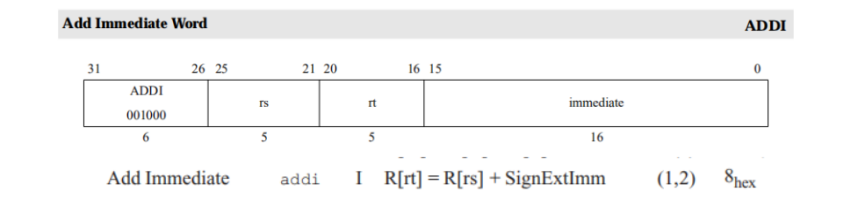
BNE：

**注意：BranchAddr根据讲解，就是C寄存器中的值，是在PC+4过程中同时进行的立即数拓展后逻辑左移2位（因为指令32位，字节寻址的话一定是0、4、8这样的数，所以逻辑左移两位补两个0保证寻址正确）然后和地址PC+4（第一节拍中完成了PC+4更新了PC）相加存入C中的，这是每个指令的译码阶段都要做的，只不过用不用取决于分支指令的结果,和分支选择指令**

分析：

第一个节拍：A寄存器和B寄存器锁存数据，产生AluSrcA和AluSrcB信号进行A寄存器、B寄存器数据传入，同时ALU的=信号产生，我们根据BEQ和BNE同时配合=信号设计控制信号输出电路（这个电路是为了若满足指令条件提前产生PC写信号），根据这个指令我们也知道PC的输入来源多了从C寄存器的值，增加控点PcSrc，同时根据结果判断PC使能端是否有输入，是否更新PC。

ADDI：

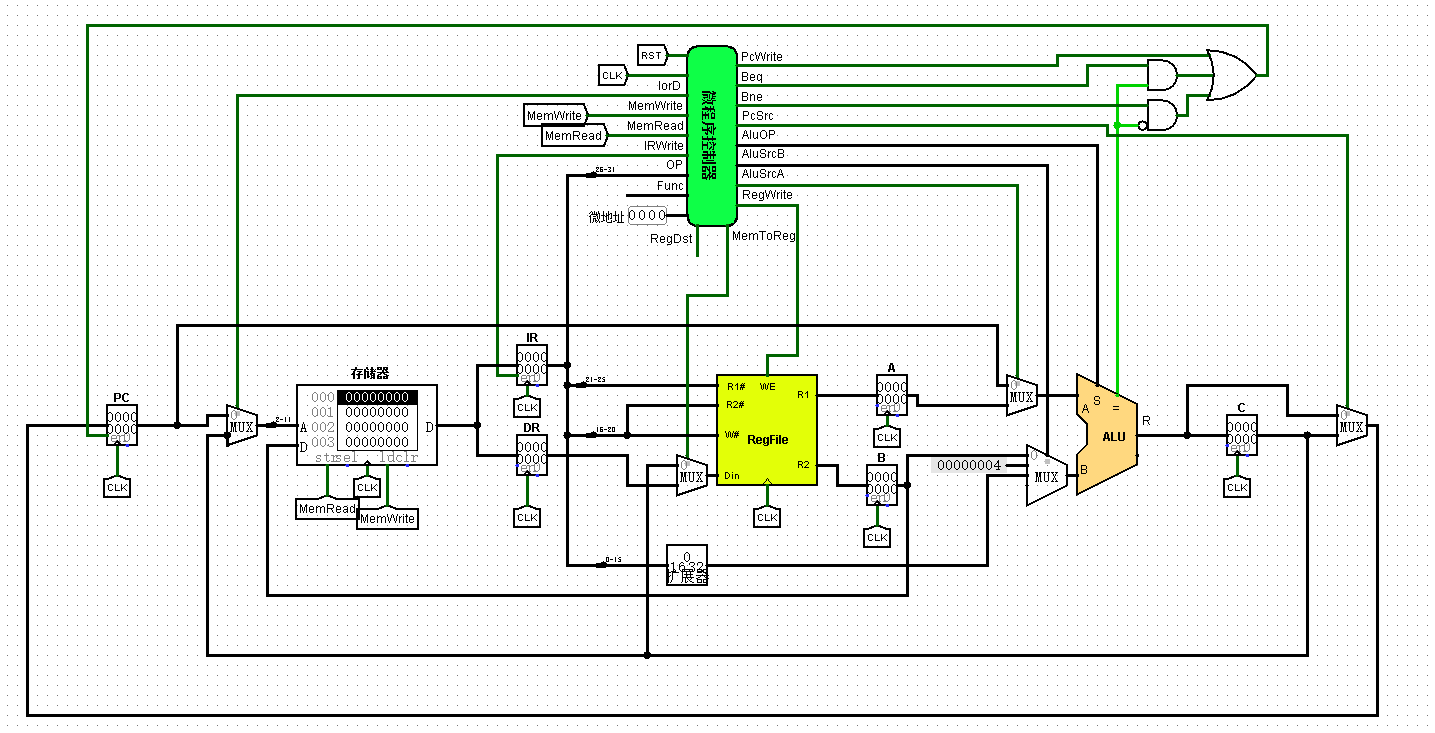


分析：

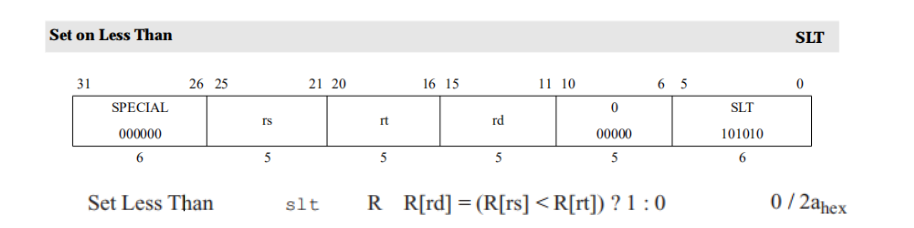
第一个节拍：寄存器A保存数值，第二个节拍：AluSrc、AluSrb、Aluop产生相应信号，计算结果。

第二个节拍：寄存器C锁存数据，输出到Din端口，同理，我们添加控点MemToReg

第三个节拍，RegWrite信号产生写入数据



* 1. R指令：SLT、ADD、SYSCALL

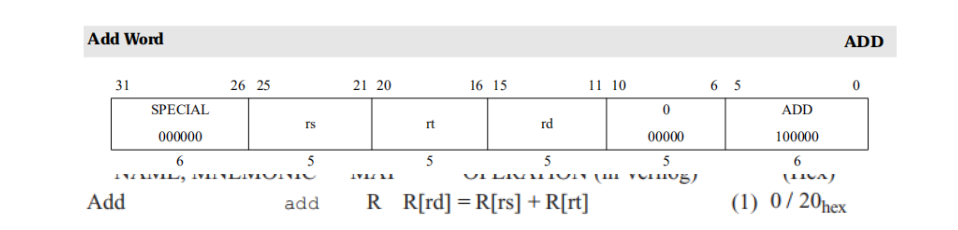
SLT：

分析：由于R指令，我们再分路一条11-15位，作为W#输入，根据原则，我们增加控点RegDst，且增加Func分路作为控制器输入。

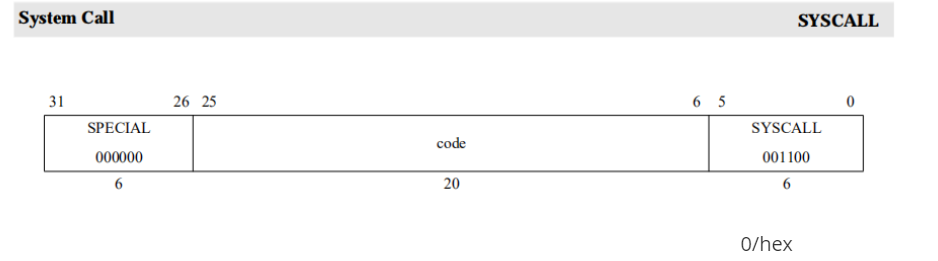
第一个节拍：RegDst信号产生，R1、R2数据锁存到A、B寄存器

第二个节拍：AluSrcA、AluSrcB、AluOP信号产生，启动ALU中设计好的判断电路并且输出，若符合则为1若不符合则为0.数据锁存到C中

第三个节拍：第四个节拍：产生MemToReg信号和RegWrite信号写入寄存器

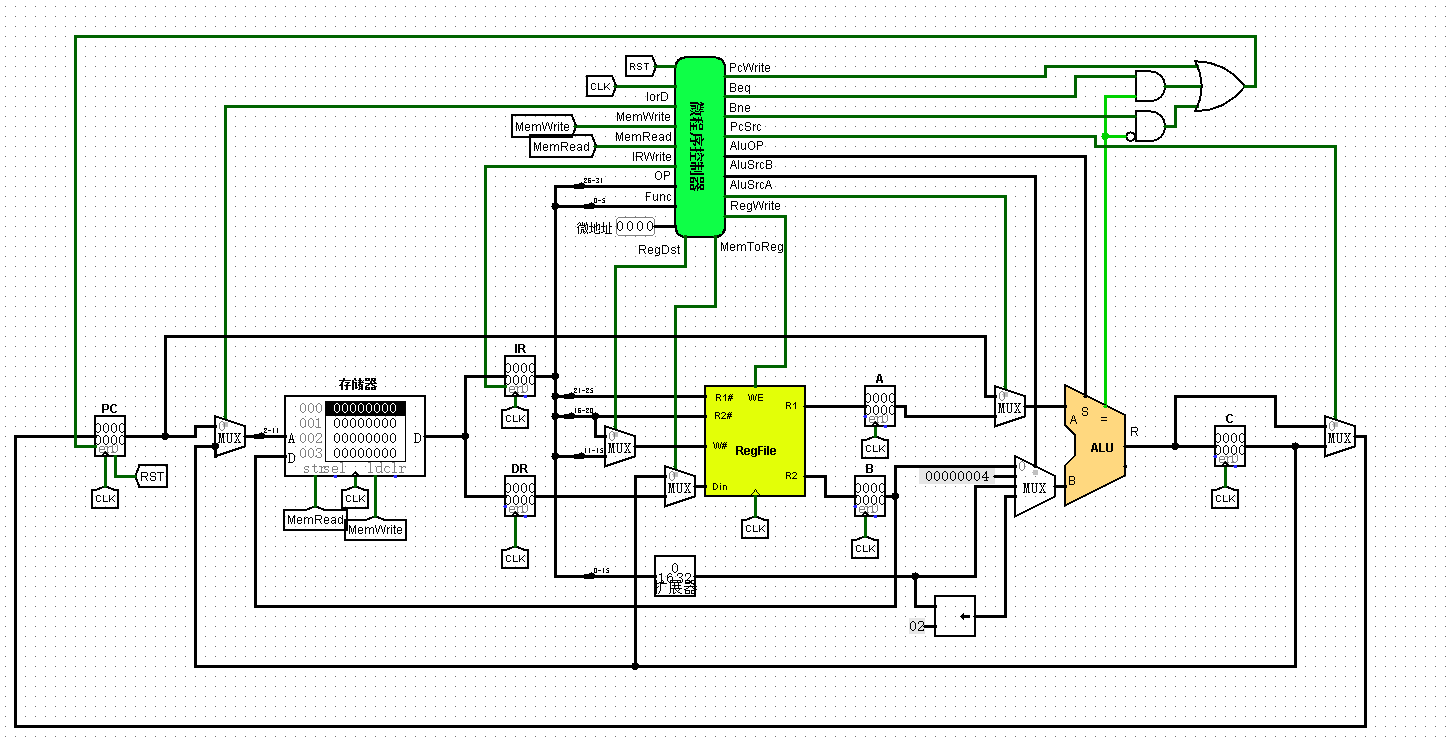
ADD:

同SLT只不过ALU启动为加法

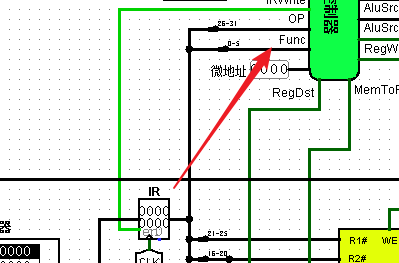
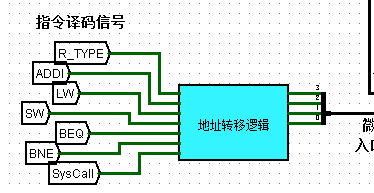
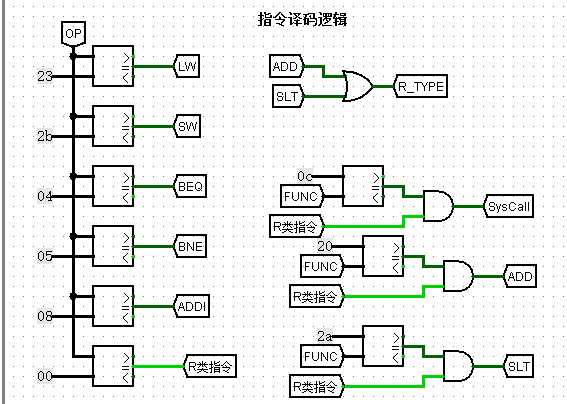
SYSCAL

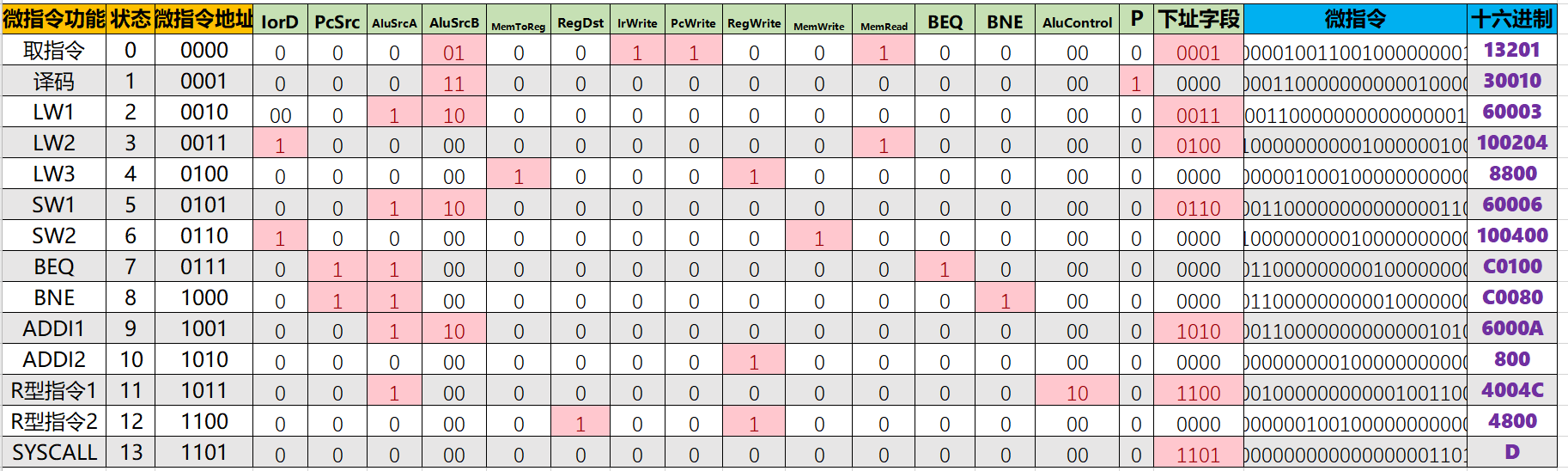
根据Func和OP字段判断到SYSCAL直接进入停机阶段

注意立即数拓展后需要逻辑左移2位，之前忘记了现在在补充说明在前面，最终通路完成

其他补充：我们的分析是取指令+各种指令执行，而MOOC上是（取指令、译码）+执行，

指令设计：

1. 首先，是微指令程序中第一个微指令是取指令信号：从而一开始启动，就开始取命令到IR寄存器中，IR寄存器的数据经过数据通路分线，产生CU的输入信号OP和FUC
2. 第二部分，通过输入的OP和FUC字段输入指令译码信号，本来是需要使用译码器的，但是由于我们只有8条MIPS指令，且是使用LOGISIM进行过程性建设，所以只需要在MIPS指令集中查找对应的MIPS指令代表的OP和FUNC字段，通过比较器比较相等即可产生译码信号，根据不同的译码信号通过地址转移逻辑进入相应的指令微程序入口（这里我们的地址转移逻辑还未设计,因为我们需要根据控存中的实际微程序地址来设计正确的转移信号）
   1. 分析8条指令，对每条指令需要的每个时钟周期进行分析，得出每个指令在每个时钟周期需要产生的控制信号，填入微指令自动生成EXCEL表格，生成16进制的各个指令对应的微程序



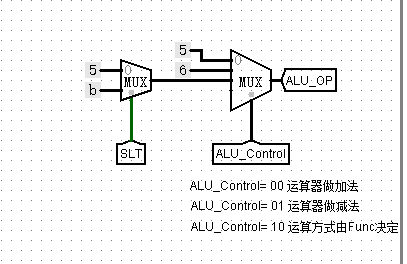
注意：

①P：为0代表通过下址字段寻址找下一个微指令，为1代表通过地址转移逻辑得到的地址寻找微指令，只有在译码阶段的下一个指令我们才设P=1从而根据地址转移逻辑得到相应微程序入口，其他的都是设计好了的通过下址字段寻址，保证顺序执行。

②微指令顺序可以不同，我们这样安排是美观且方便人阅读和设计，但是需要注意相应的寻址设置需要进行准确设置，保证找到指令的下一个时钟周期需要的微指令产生正确的控制信号输出

③一般所有指令的过程都是 取指令——>译码——>通过地址转移逻辑进入指令对应微程序的入口地址——>执行微程序，微程序的最后一条指令下址字段设置为取指令地址——进入下一个指令。但是SYSCALL停机指令则是下址字段设置为自身，进行不断调用自身来进行模拟停机功能。

④关于ALU\_OP的控制信号，也就是ALU的S功能选择端口的信号，我们不是直接输出，而是通过微指令中的ALU\_Control信号，决定使用ALU的哪个功能。这样设计才能够使得R型指令的微程序成为1个，根据不同的R型指令具体译码信号，自动产生该具体R型指令下需要的ALU操作控制信号，不必根据不同的指令专门设计只是更改了ALU操作的微程序。

通过分析，我们的8条指令其中除了SLT操作，其他需要使用的都是加法，而我们的ALU加法控制端设置为00，所以一般情况下，也就是ALU\_Control无信号ALU默认为加法器工作。若为10则进入R指令的ALU信号选择。若输入SLT信号则选择比较功能否则就是加法功能

1. 设计好各个指令的微程序后将16进制微程序码存入控存。这样我们就能够根据设计好的不同指令的微程序入口设计地址转移逻辑:

为不同指令填写相应入口地址生成相应的S0到S3产生的逻辑使用LOGISIM的分析电路功能完成转移逻辑设计



这样我们的微程序CU设计就完成了。

在完成微指令设计的情况下，硬布线实验其实很好完成，因为将控制器视作黑盒，其实微程序和硬布线实现就是黑盒的内部工作原理，我们不关心内部原理下，只看效果其实两者是相同的，所以微程序CPU完成的数据通路可以直接使用到硬布线的通路上，所以接下来主要工作就是完成硬布线的CU就可以了。

     再进一步，在本次实验中，老师为了减少我们的工作量，状态转换成输出信号的组合逻辑电路，实际上用了和微程序相同的控存代替，同时也意味着我们先前设计的微指令全部可以照搬过来使用代表“组合逻辑生成的信号”，指令译码逻辑更不用说，直接相同，ALU控制器逻辑也可以照搬。

     不断简化下，实际上硬布线设计主要就是设计状态机，而我们之前设计微指令使用的生成表直接拿过来用，不必再次分析每个节拍产生什么信号了，同时使用文件中给的硬布线状态生成表，又一次大大减少工作量，我们只需要分析之前设计的微指令序列，根据状态填表自动得到次态生成信号表达式，再利用LOGISIM的自带电路分析功能，直接生成电路。硬布线就完成了。

