P3（Logisim单周期CPU）

▼ Contents

▼ Problems

- [x] ~~怎么用MIPS结合ROM与RAM测评啊啊啊啊~~

《*Digital Design and Computer Architecture*》救我

整体结构

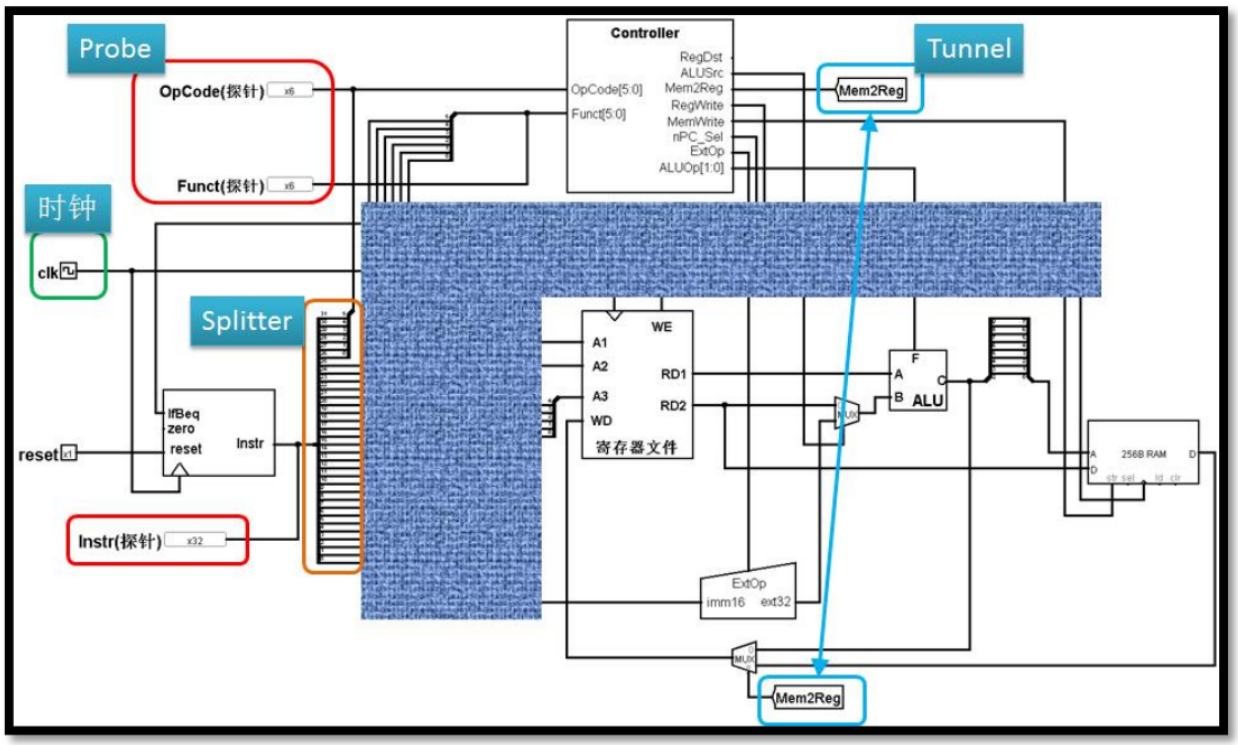
概述

将使用 Logisim 开发一个简单的 MIPS 单周期处理器，并使用 Mars 自行编写测试程序，验证 CPU 设计的正确性

基本思路

设计的 CPU 将包含 Controller（控制器）、IFU（取指令单元）、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、EXT（位扩展器）等基本部件，通过 MUX、 Splitter 等内置器件组合连接成数据通路

可能参考图：



模块规格

模块规格是设计文档的重要部分，它包含了 CPU 各个模块的**端口说明**与**功能定义**。一份好的模块规格可以让其他人快速理解该模块的功能并加以实现。这就相当于一份 CPU 重要部件的“说明书”。

在模块规格这个部分我们不直接给出详细的端口说明与功能规定，仅给出简略的要求，希望同学们在使用 Logisim 搭建 CPU 过程中，自行完成相应的设计，**清晰阐明所有模块的功能，并给出关键模块（如 GRF、DM ）的各端口定义及内部逻辑实现。**

设计要求

使用Logisim开发一个简单的MIPS单周期CPU，总体概述如下：

1. 此CPU为32位CPU
2. 此CPU为单周期设计
3. 此CPU支持的指令集为：{addu, subu, ori, lw, sw, beq, lui, nop}
4. nop机器码为0x00000000
5. addu, subu 不支持溢出

以下说明均从功能的角度出发，因为具体的端口声明可以在我的设计文档中找到

模块要求

- IFU（取指令单元）：内部包括 PC（程序计数器）、IM(指令存储器)及相关逻辑。
  - PC 用寄存器实现，应具有**异步复位**功能，复位值为起始地址。
  - **起始地址：0x00000000。**
  - IM用 ROM 实现，容量为 32bit * 32。
  - 因 IM 实际地址宽度仅为 5 位，故需要使用恰当的方法将 PC 中储存的地址同 IM 联系起来。
- GRF（通用寄存器组，也称为寄存器文件、寄存器堆）
  - 用具有写使能的寄存器实现，寄存器总数为 32 个，应具有**异步复位**功能。

    - **0 号寄存器**的值始终保持为 0。其他寄存器**初始值（复位后）均为 0**，无需专门设置。
- ALU（算术逻辑单元）
    - 提供 32 位加、减、或运算及大小比较功能。
    - 可以不支持溢出（不检测溢出）。
- DM（数据存储器）
    - 使用 RAM 实现，容量为 32bit * 32，应具有**异步复位**功能，复位值为 0x00000000。
    - **起始地址：0x00000000**。
    - RAM 应使用双端口模式，即设置 RAM 的 **Data Interface** 属性为 **Separate load and store ports**。
- EXT：
    - 可以使用 Logisim 内置的 Bit Extender。
- Controller（控制器）
    - 使用与或门阵列构造控制信号，

控制器设计

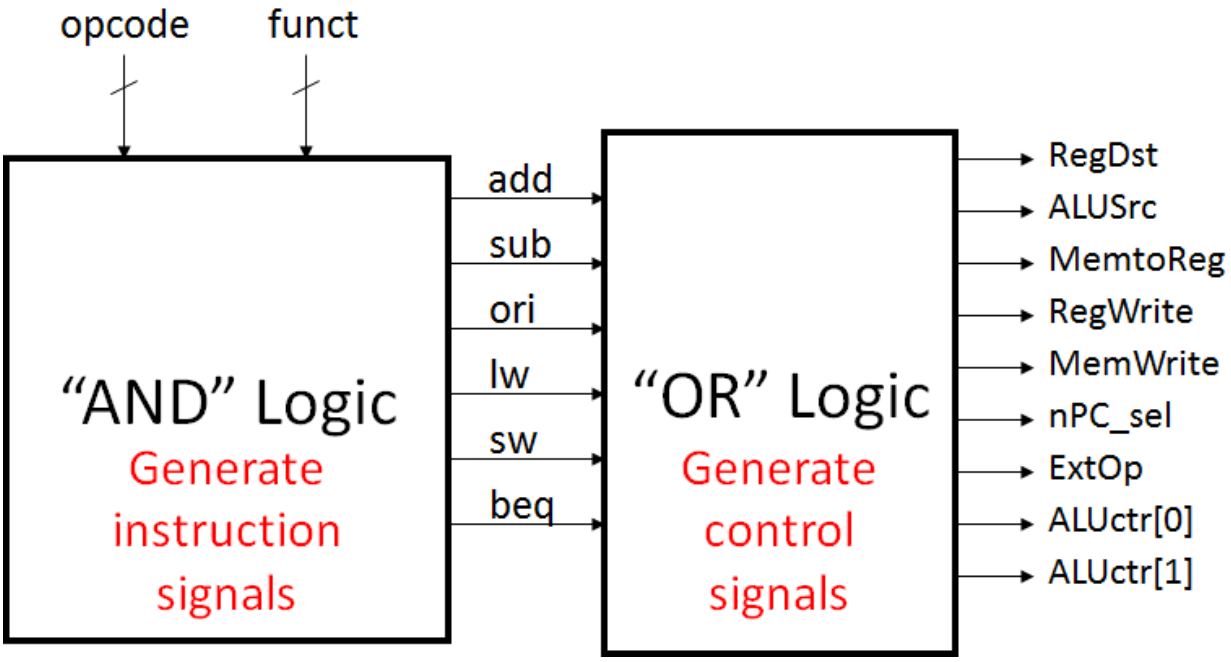
控制器设计

控制器的设计，从最基本的层面来说，是一个**译码**的过程，将每一条机器指令中包含的信息，转化为给 CPU 各部分的控制信号（ RegDst， ALUSrc 等等）。

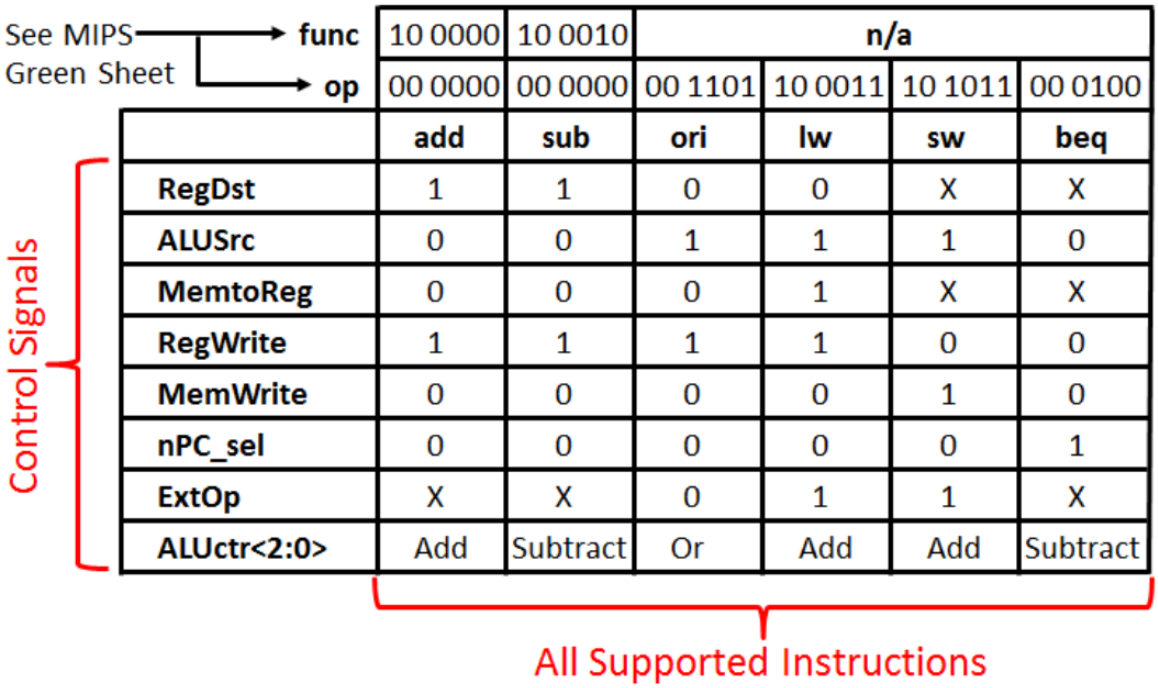
到了实践层面，如何让解码过程变的工程上可实现，是一个重要的问题。最为直接的办法就是生成真值表，但是这样在实际操作中不具有可扩展性与易调试性，指令数增加时非常容易导致 Bug 产生。

于是使用下面一种可行的办法之一

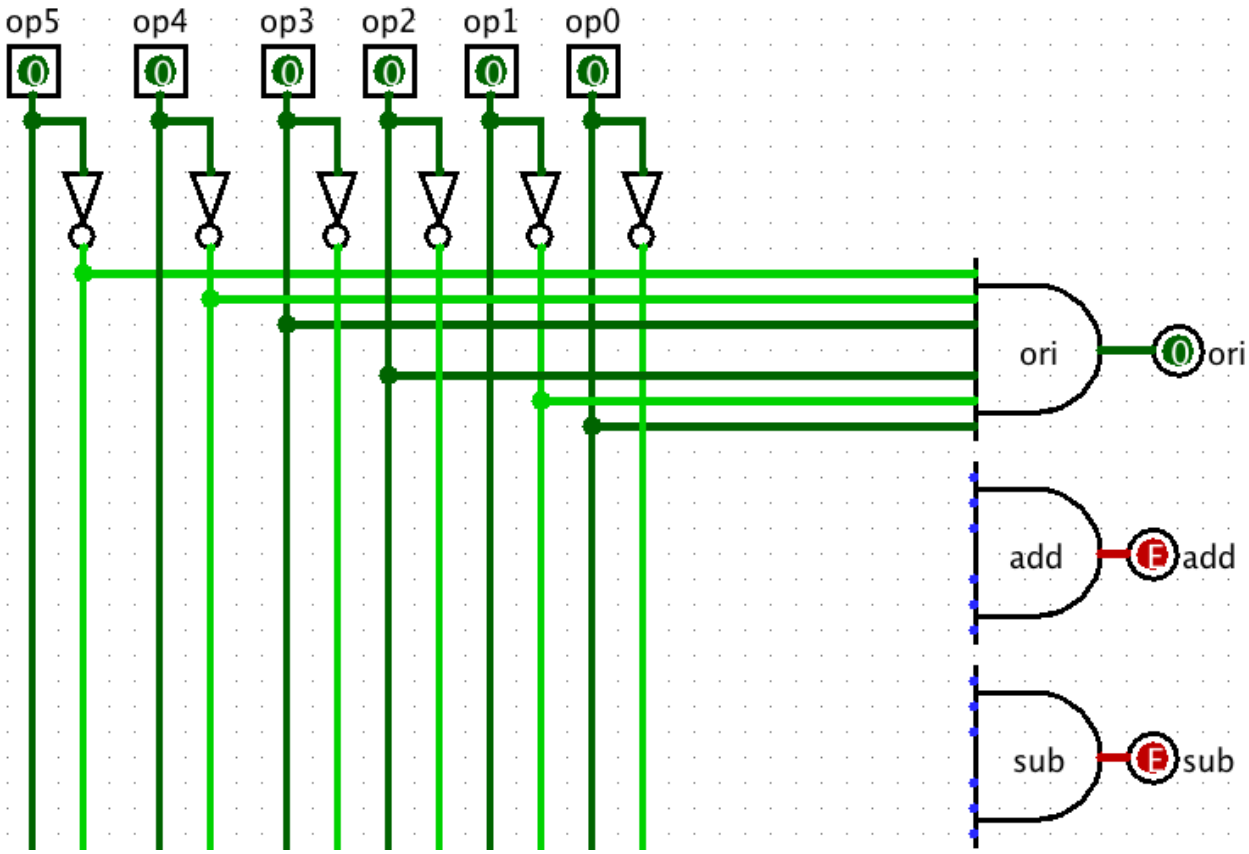
把解码逻辑分解为**和逻辑**和**或逻辑**两部分：和逻辑的功能是**识别**，将输入的机器码识别为相应的指令；或逻辑的功能是**生成**，根据输入的指令的不同，产生不同的控制信号。这种拆分使得两部分逻辑目的明确，这是一种朴素的**抽象**与**模块化 要在**其他的领域也加以借鉴

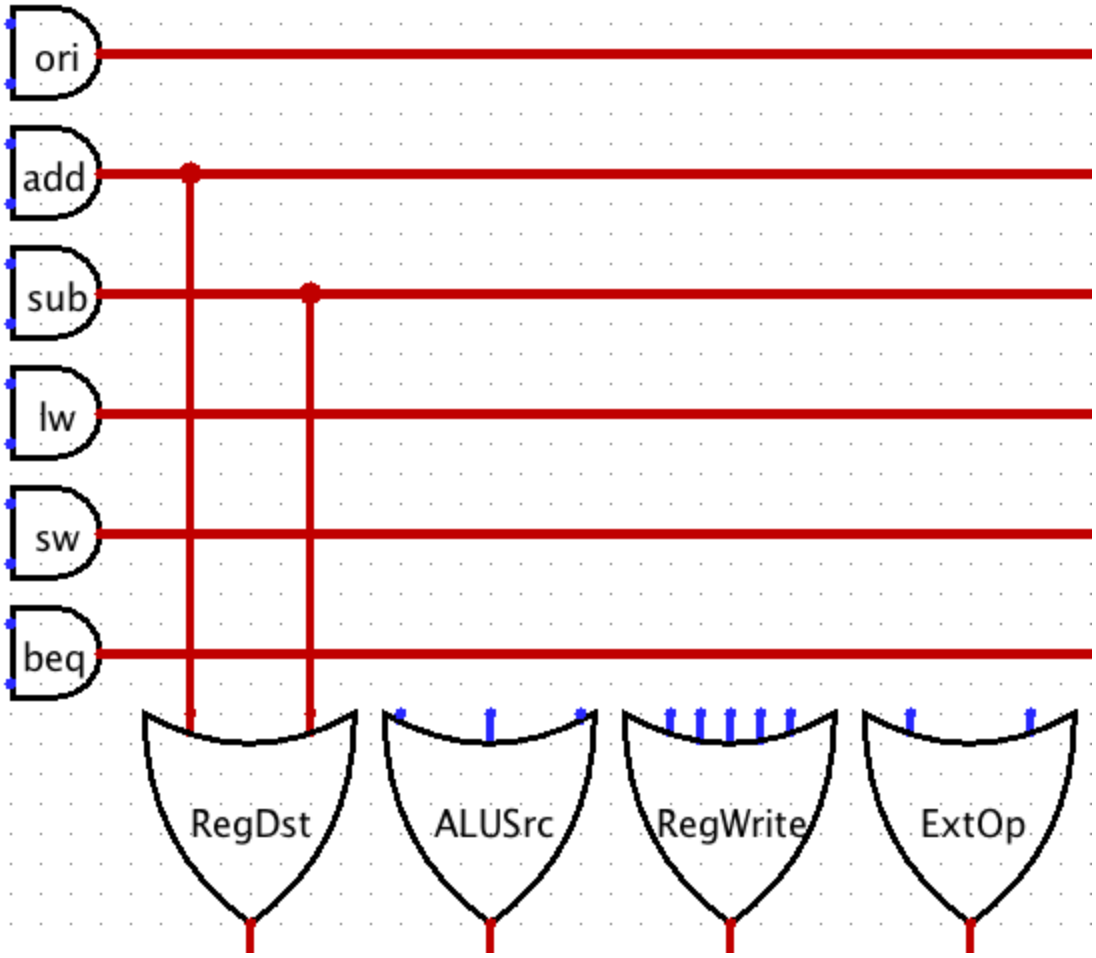


而在设计这两套逻辑的过程中，和逻辑是非常自然的。而或逻辑则需要我们建立从指令到控制信号的映射，为了避免错误的产生，我们希望使用真值表来完成相应的设计任务，并希望通过真值表可以简化相应的逻辑。一个典型的真值表如下图：



下图是上述逻辑在 Logisim 中的电路具体实现样例——**与或门阵列**（并不完整），当然你也可以有自己的电路设计，比如使用 Tunnel 来简化布线、对指令进行合理的分类建模等方案。


op5
op4
op3
op2
op1
op0
ori
ori
add
add
sub
sub



文档撰写建议

- 描述控制器的具体电路逻辑，并阐释你如何将控制器设计得更简洁且有条理。
- 建立一张表来说明各个控制信号的功能。

测试程序设计

要点

- 保证测试中所用的指令不超出**CPU支持的范围**

- 需要全面**测试**每条指令，不仅指能实现的指令，也指每个指令的各个方面，如果是加法的话，就要把正数之间相加，负数之间相加，正负相加的各种情况

教程部分就这么戛然而止了，对于每个模块的设计并没有具体的要求。感觉像是每个模块都可以对应一个P0组合逻辑的问题，目前存在的在于不会测试，先自顶向下搭一个静态的出来吧

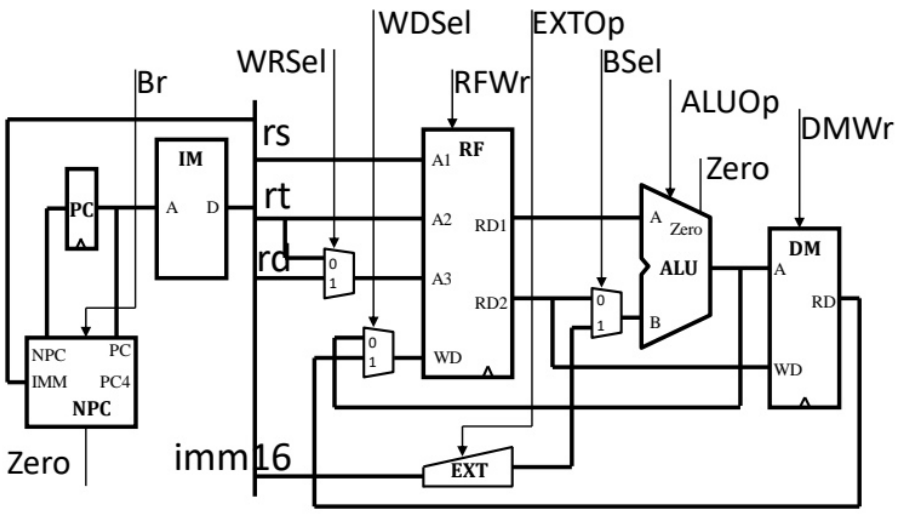
大体框架

具体的成果介绍看设计文档好了

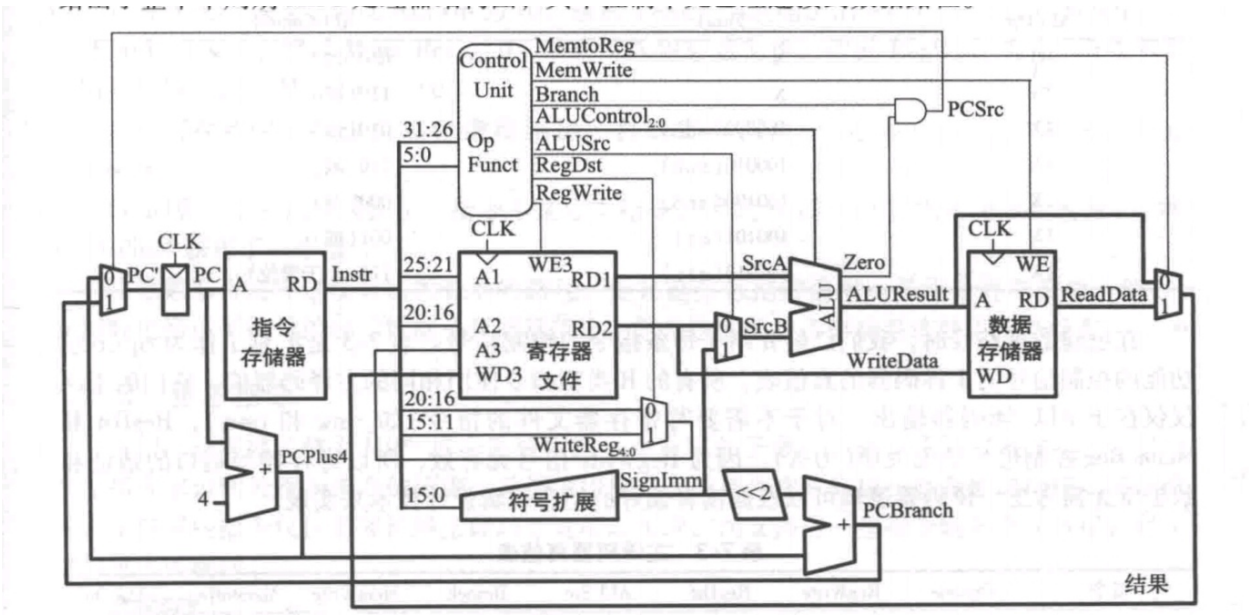
设计文档

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c777f57c-e777-4718-9de3-d93187ee4488/Logisim单周期CPU.docx

课上说明的可以执行，add,sub, ori, lb, lw, sw, beq



或者按照黑书，大体相同，在DM上多了个MemtoReg的判断信号，用于处理R型指令



按照黑书为框架搭建吧(感觉上就多了一个EXT可以是0扩展的其余大致一样)

模块分析

ALU

根据CPU需要实现的指令来确定自己的ALU要做哪些操作。

比如加减运算等，

通过一个ALUop来“告诉”CPU要对输入的数干什么，然后输出。

感觉应该是最简单的了吧😣

DM

也是难度较小的一个模块，但有以下几个思考点

1. ROM，RAM与Register的选择

内存是需要有读取和写入两个操作的，而ROM只有读取，out

在一个周期内，DM只会有读取和写入操作中的一个，因此大可不必上GRF那般的Register，加上DM空间容量也大，上GRF不够现实，因此选择**RAM**

2. 寻址

问题在于MIPS和Logisim寻址的区别的转换。MIPS地址按字寻址，而RAM是是按字节，且MIPS中的地址是32位，Logisim是5位，即MIPS中的PC+4对应的是RAM中Address+1，如果地址从0开始的话，那么MIPS中就是0， 4， 8，……，而RAM中就是0， 1， 2， ……，正好是4倍的关系，再结合二进制下有关2^n的倍数等同于右移，因此我们要取Address的2-6位

（或许也可以这么想，+4对应成二进制就是+100，因此第0，1位没啥影响，对序列数产生直接影响的从第2位开始）

EXT

好吧我错了，这应该才是最简单的模块，用两个bit_extender(对应0扩展和符号扩展)，一个MUX（以EXTOp为选择信号），就愉快的搭完了

GRF

似乎明白了P0搭GRF的奥义，搬过来照常用就好了

IFU

内部包括 PC（程序计数器）、IM(指令存储器)及相关逻辑

- IM

有之前分析可得，IM只有读取的操作，因此选用ROM足够了

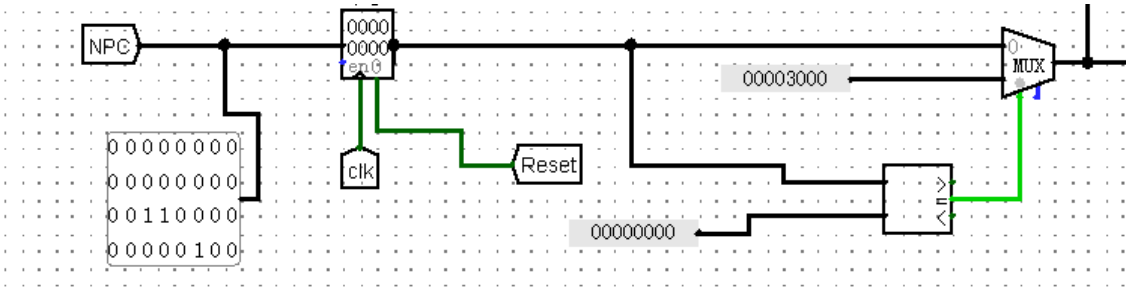
主要是根据PC值取指令，这边具体有两个需要注意的点

1. 和DM一样，针对PC+4，取2-6位作为ROM的地址
2. 异步复位的问题

   Logisim复位是复位到0x0000_0000加上地址确实也是从0x0000_0000开始的，问题不大

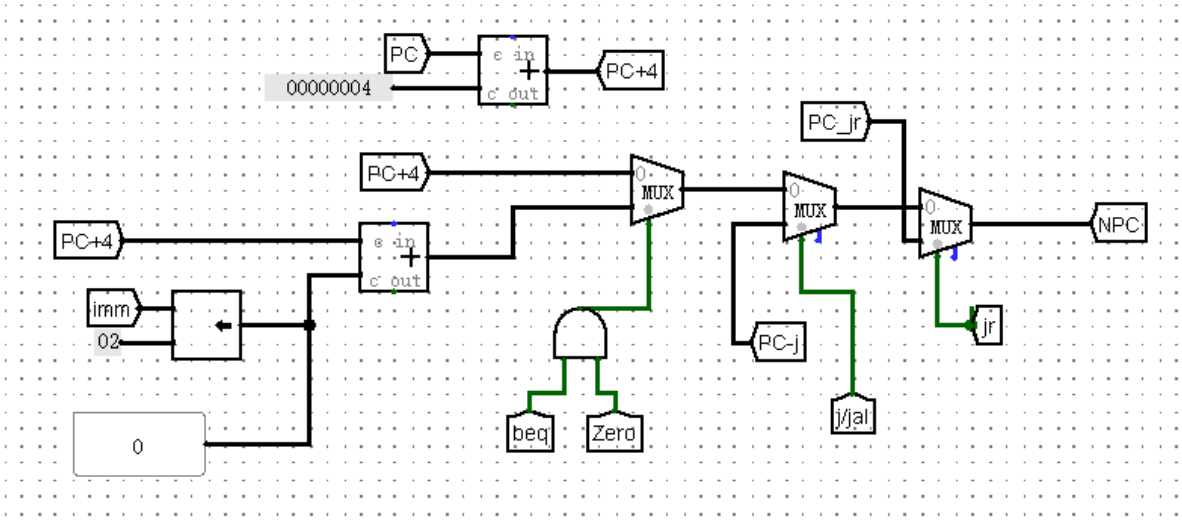
   但是，为了和MIPS有一致性，我的写法是复位到0x0000_3000，然后再统一减去3000开始

   这时需要一个比较器和MUX



- NPC:

  正常PC+4，其余可以根据指令的特殊性进行跳转，关键在于用MUX进行一次条件判断



Controller

这个在先前的教程中已经较为完备的展现了，就是用与或门阵列判断指令，再根据每个指令所需要实现的signal进行反馈

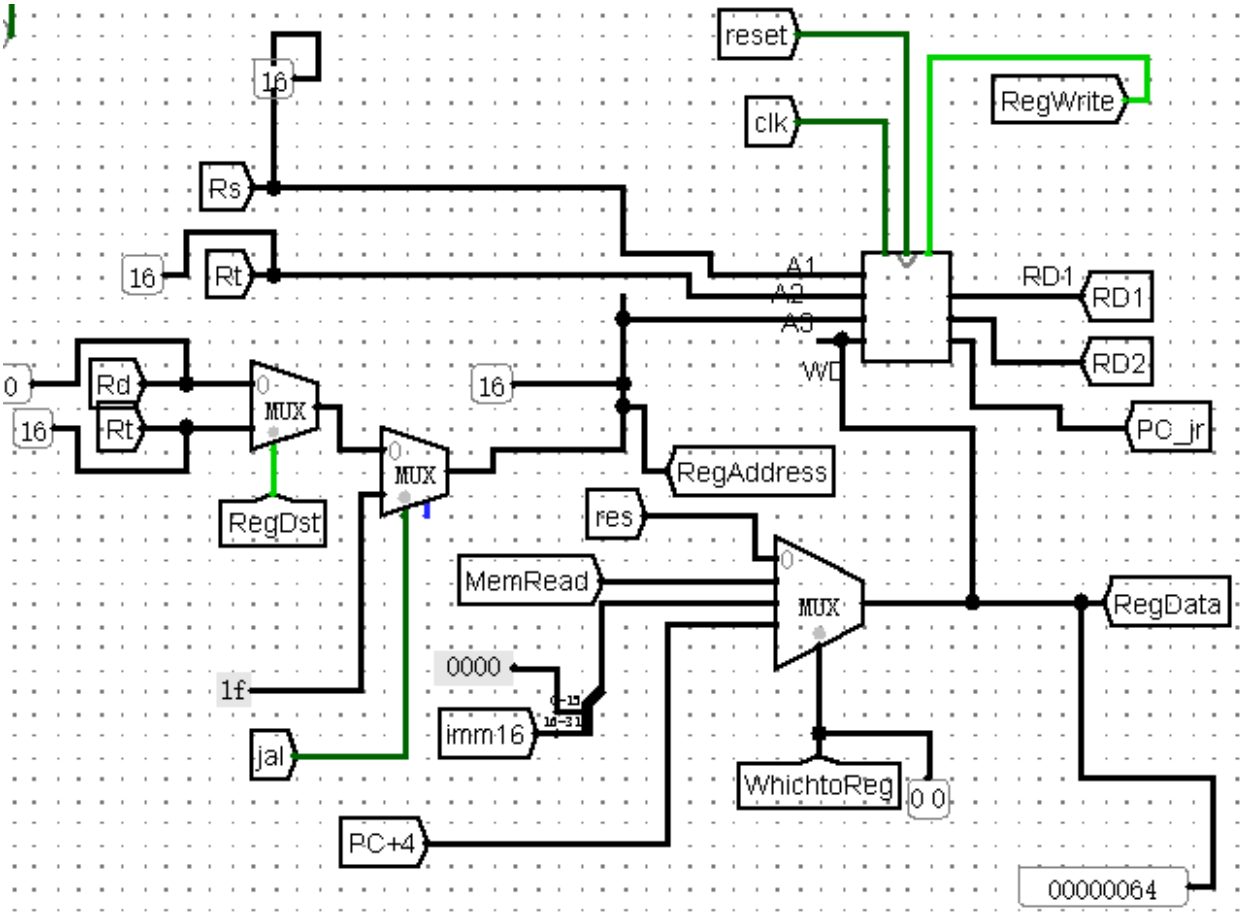
一些有意思的指令设计

有意思针对的数据通路，与一些有意思的运算处理无关

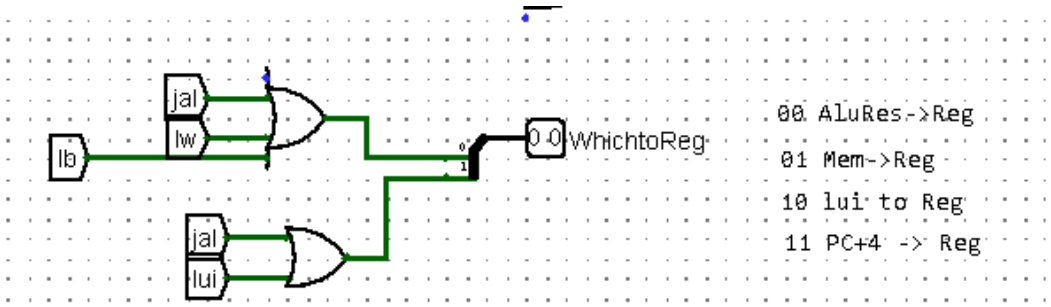
j/jal，jr

每个周期都按照RTL进行计算，再用mux判定

对于jal，可以再利用MUX对GRF进行修改



whichtoReg决定了写入A3的内容



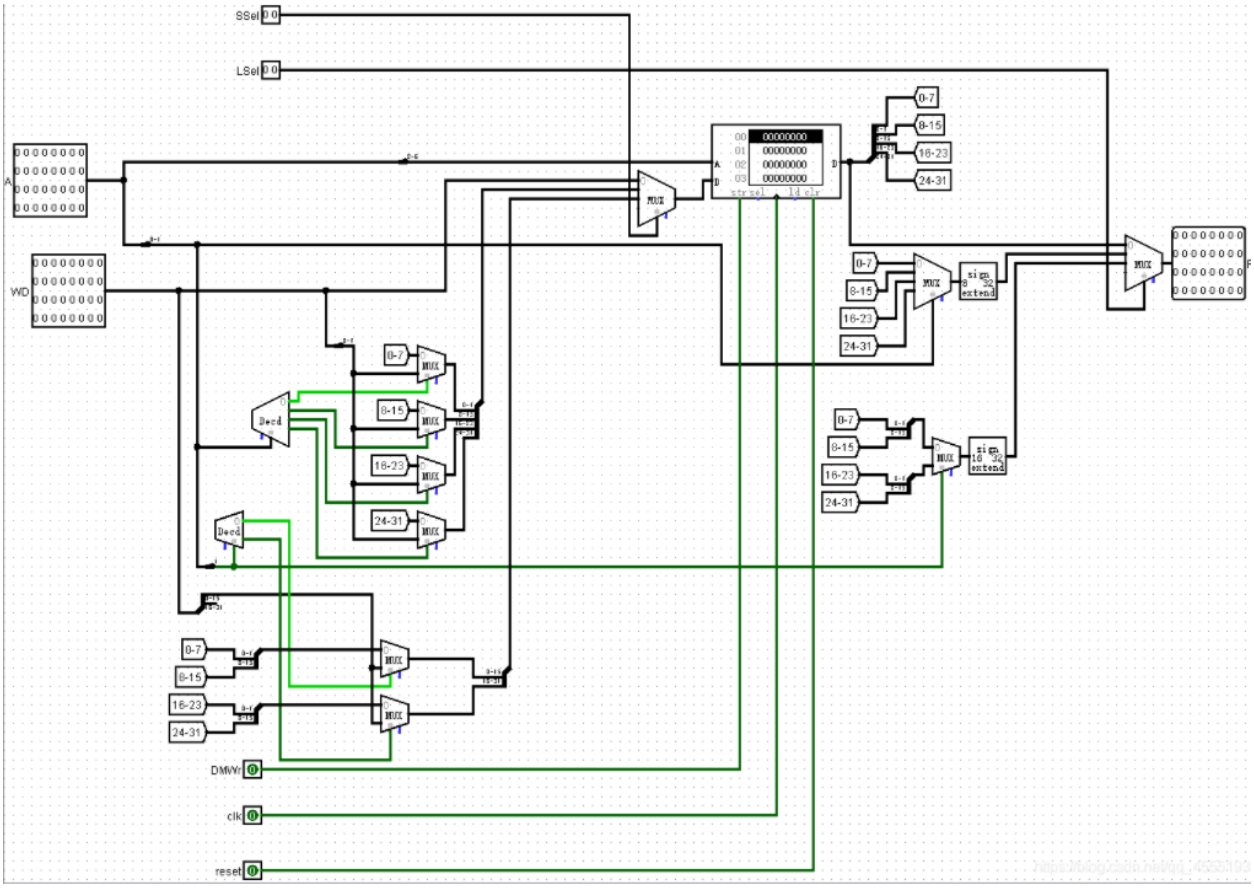
做到P4发现自己jr搭错了

是自己对指令理解的问题, jr的意义应该是 `PC = GPR[rs]` ~~我以为是~~ ~~`PC=GPR[31]`~~

sb，sh/lb，lh

Key：我们得到的32位地址的2-6位可以提供内存中的某一个地址，而后0,1位可以用来判定具体地址的部分位，如sb/lb 对应的[0,7] [8,15], [16, 23], [24, 31] sh/lh对应的[0-15], [16, 31]

同时，因为sw/sh 需要再原数组基础下，挑一部分出来修改，因此我们需要对传入的选择信号选择WD的一部分和原有的数据拼接，这就需要译码器的使用，根据传入的addr的后面两位选择需要拼接的WD[7:0] 或 [15:0]



👇 附上自己搭建的CPU把

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b03718b0-7a6b-46dd-8ae8-64feb3563e63/Cpu.circ

P3 课上

和往年类似，一个跳转一个运算一个访存

bezal

if GPR[rs] == 0:

    PC = GRF[rt]

    GPR[$31] = PC+4;

else

    PC = PC+4

把我杀死的一个题，因为condition写的太离谱了

可以考虑本身指令附带条件的话，再Controller结束后将两个条件利用与门更新一下，可以确保自己不会迷糊

slo

将一个数左移并补1，看到最百花齐放的做法

看到比较多的好像是

1. 先弄出s个1，然后利用bit-extender（0扩展）弄到32位，然后将正常左移得到的数进行或运算（利用或运算保持非0值的性质）
2. 将原来的数+1，后左移s位，再扣掉1 （看到那么多1会有一个+1的想法吧哈哈哈哈）

lwor

印象中是lw改了一点

GPR[rt] = memory[GPR[base] + offest + rt*4]

也没啥，算address加一个rt*4就好了，但确实题目讲的有点不够明白，这里的rt指的是寄存器的编号，就真的是Instr[20:16]的东西