P3 Logisim 单周期CPU

一、模块规格

1. PC 程序计数器

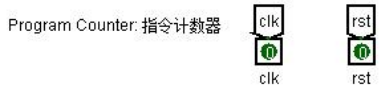
端口定义

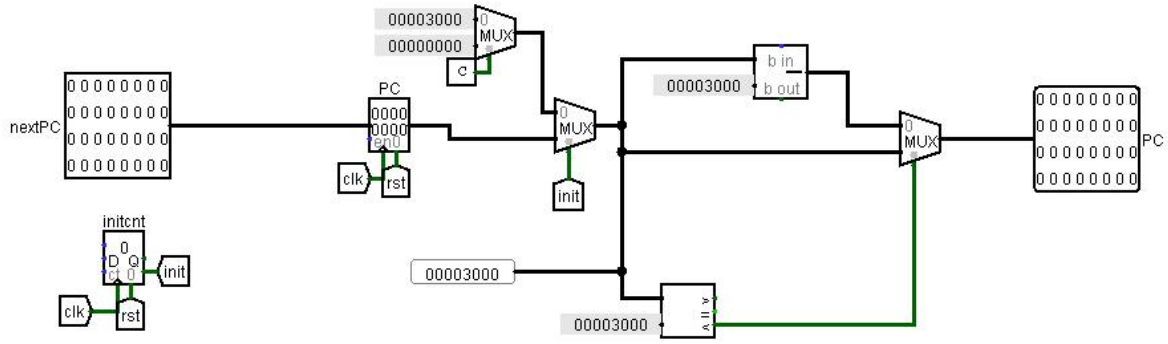
端口名	位宽	方向	信号作用
clk	1	Input	时钟信号
rst	1	Input	异步复位信号
nextPC	32	Input	下一周期的 PC
PC	32	Output	当前 PC 值

功能描述

序号	功能名称	功能描述
1	复位	当 rst 信号为高时异步复位
2	下一 PC 值	不在复位状态且 clk 上升沿时， PC 值更新为 nextPC 的值

电路图





2. NPC 下一指令地址计算

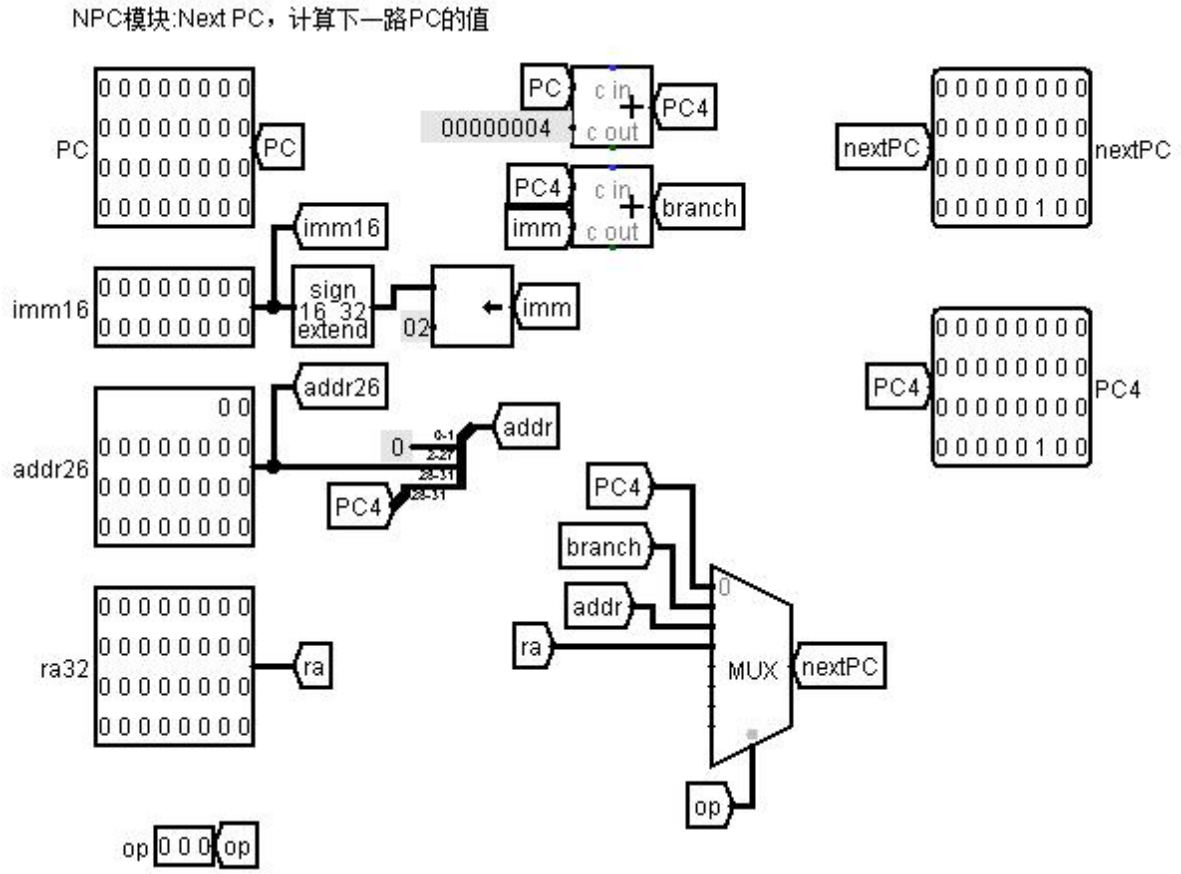
端口定义

端口名	位宽	方向	信号作用
PC	32	Input	当前 PC 值
imm16	16	Input	I 型指令中的立即数
addr26	26	Input	J 型指令中的立即数
ra32	32	Input	$rs 寄存器中的值
op	3	Input	下一 PC 值的选择信号
nextPC	32	Output	应执行的下一条指令的 PC 值
PC4	32	Output	当前 PC 值 + 4

功能描述

序号	功能名称	功能描述
1	输出下一条指令的 PC 值	PC4 信号始终输出当前 PC 值 +4
2	选择下一周期应执行的指令地址	当 `op == 0` 时，顺序执行，`nextPC = PC + 4` 当 `op == 1` 时，由分支指令跳转，`nextPC = PC4 + (sign_ext(imm16) << 2)` 当 `op == 2` 时，由 `j` 或 `jal` 指令跳转，`nextPC = {PC4[31:28], addr26, 2'b0}` 当 `op == 3` 时，由 `jr` 或 `jalr` 指令跳转，`nextPC = ra`

电路图



3. IM 指令存储器

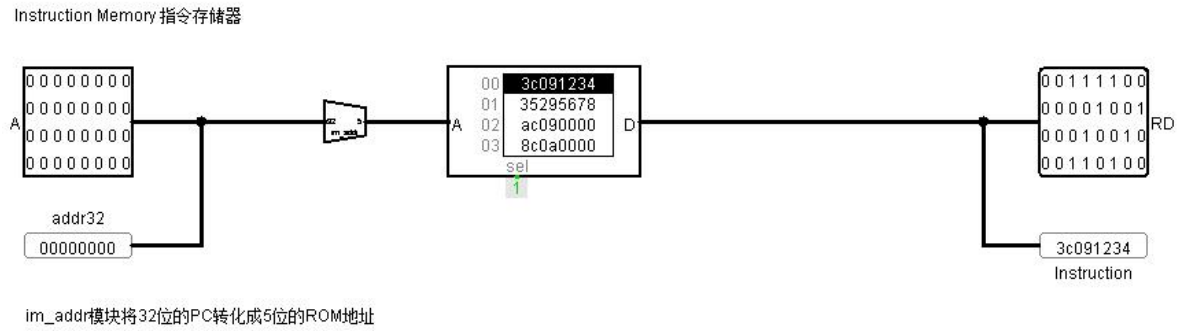
端口定义

端口名	位宽	方向	信号作用
A	32	Input	32位输入地址
RD	32	Output	当前地址读出的指令

功能描述

该模块内含有 ROM 存储器，用于存放程序指令。将 `A` 输入中的 32 位地址通过内部的一个模块 `im_addr` 截断成 5 位，从 32bit * 32 的 ROM 中取出指令。

电路图



4. GRF 寄存器文件

端口定义

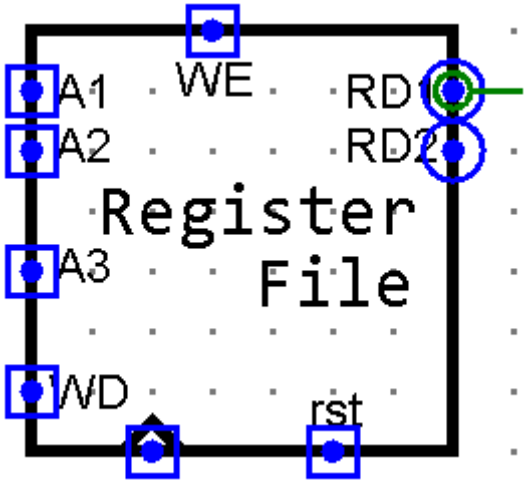
端口名	位宽	方向	信号作用
clk	1	Input	时钟信号
rst	1	Input	异步复位信号
WE	1	Input	寄存器写使能信号
A1	5	Input	5位地址信号，选择一个寄存器，将值输出到 RD1
A2	5	Input	5位地址信号，选择一个寄存器，将值输出到 RD2
A3	5	Input	5位地址信号，选择一个寄存器用于写入
WD	32	Input	32 位数据输入
RD1	32	Output	输出 A1 信号选出的寄存器存储的数据
RD2	32	Output	输出 A2 信号选出的寄存器存储的数据

功能描述

序号	功能名称	功能描述
1	读取寄存器	根据 A1, A2 指定的寄存器编号读取寄存器数据到 RD1, RD2
2	复位	rst 信号高电平时异步复位所有寄存器（值变为0）
3	写入寄存器	当时钟上升沿到来且写使能信号 WE 为高电平时，将 WD 中的数据写入 A3 指定的寄存器。特别地，0号寄存器的值恒为0，无法写入。

电路图

由于本部件的内部为大量相似电路复制而成，故电路图省略。此处只放出模块外观图。

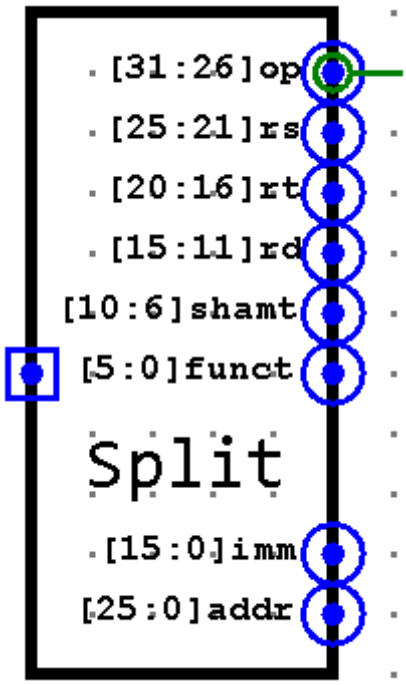


5. ISplitter 指令分线器

模块说明

将一条 32 bit 的指令根据 MIPS-32 指令格式拆分出相应的部分。内部采用 Splitter 部件实现，此处端口定义省略，仅放出模块外观图。

电路图



6. 位扩展器

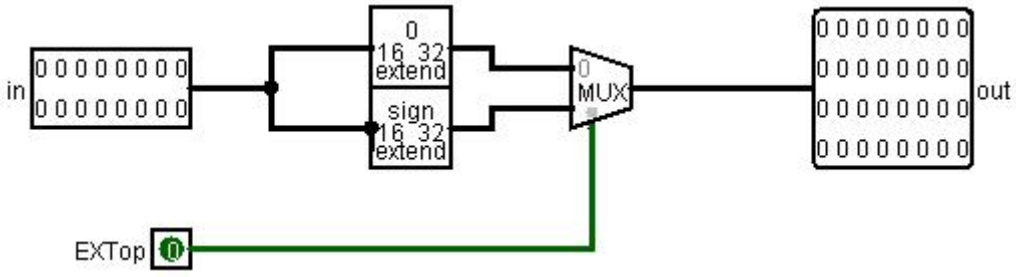
端口定义

端口名	位宽	方向	信号作用
in	16	Input	I 型指令中的 16 位立即数
EXTop	1	Input	位扩展的种类，0为零扩展，1为符号扩展
out	32	Output	扩展后的 32 位立即数

功能描述

将 16 位立即数扩展为 32 位，根据选择信号 EXTop 选择进行零扩展或者符号扩展。

电路图



7. ALU 算术逻辑单元

端口定义

端口名	位宽	方向	信号作用
A	32	Input	第一个运算数
B	32	Input	第二个运算数
op	4	Input	运算种类选择信号
Out	32	Output	两数的运算结果
zero	1	Output	零标志信号
nega	1	Output	负标志信号

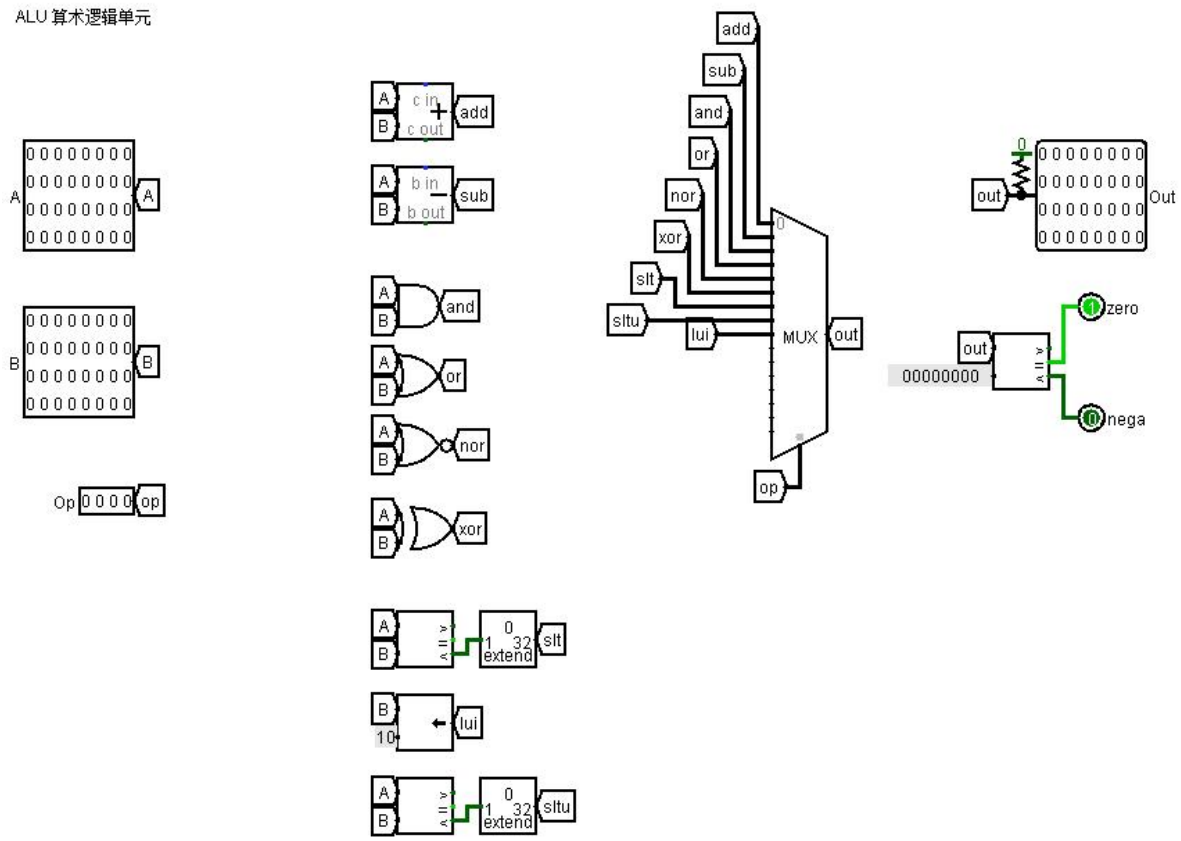
功能描述

根据 op 选择信号选择不同的运算，输出 A 与 B 进行相应运算后的结果并输出到 Out 。当 Out 为零或为负值时，zero 标志信号或者 nega 标志信号相应被置为 1。

选择信号 op 与运算种类的对应关系表：

选择信号 op 的值(十进制表示)	运算描述
0	Out = A + B
1	Out = A - B
2	Out = A & B
3	Out = A | B
4	Out = ~(A | B)
5	Out = A ^ B
6	Out = (A < B) ? 1 : 0，A B当做有符号数
7	Out = (A < B) ? 1 : 0，A B当做无符号数
8	Out = (B << 16)

电路图



8. 移位器

端口定义

端口名	位宽	方向	信号作用
In	32	Input	输入数据
sh	5	Input	位移量
op	3	Input	移位运算种类选择信号
Out	32	Output	移位运算的输出

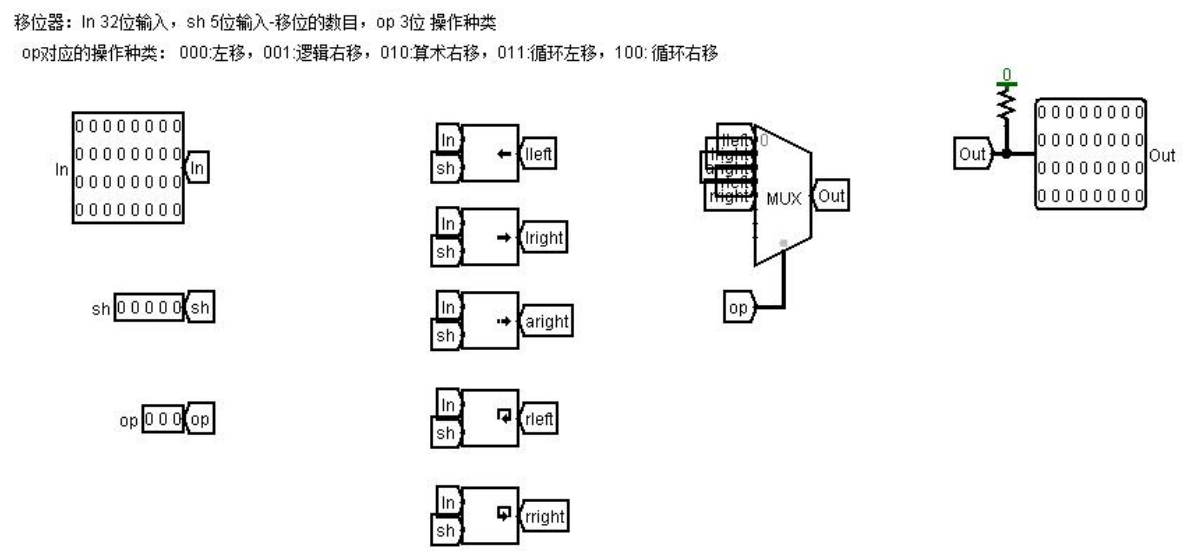
功能描述

根据 op 选择的信号不同，对输入数据 In 以偏移量 sh 进行移位运算。

选择信号与移位种类对应表：

选择信号 op 值	功能描述
000	逻辑左移
001	逻辑右移
010	算术右移
011	循环左移
100	循环右移
101, 110, 111	未使用，输出全 0

电路图



9. DM 数据存储器

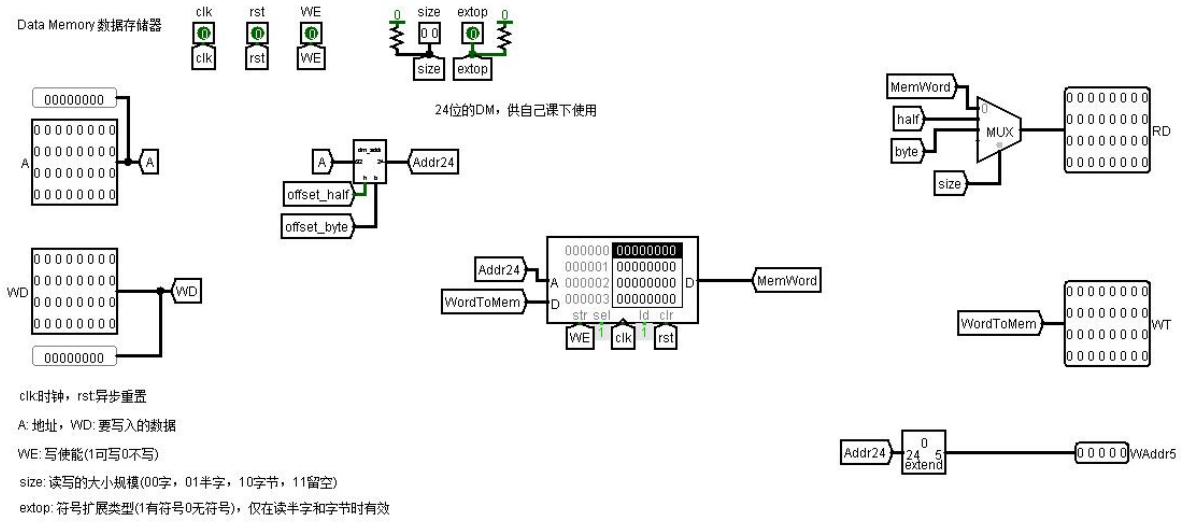
端口定义

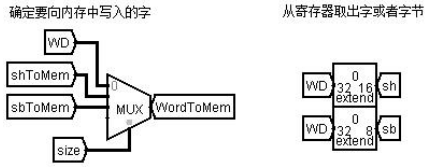
端口名	位宽	方向	信号作用
clk	1	Input	时钟信号
rst	1	Input	异步复位信号
WE	1	Input	写使能信号
A	32	Input	数据地址
WD	32	Input	准备写入的数据
size	2	Input	读写数据的规模(00: 字，01: 半字，10: 字节，11: 未使用)
extop	1	Input	读出数据时的扩展种类(0: 零扩展，1: 符号扩展)
RD	32	Output	从地址 A 读出的数据
WT	32	Output	即将写入 RAM 部件的数据（以字为单位，当 sh 或 sb 时其值与输入信号 WD 不同）
WAddr5	5	Output	评测机要求输出的 5 位写入地址

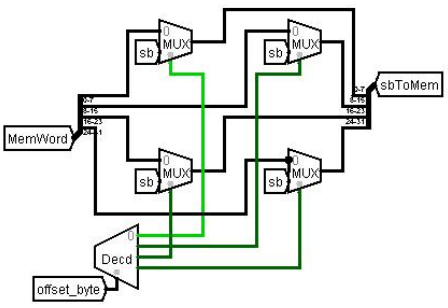
功能描述

序号	功能名称	功能描述
1	读取	根据输入地址 A 与 控制信号 size, extop 从 RAM 中读取数据。
2	写入	根据地址 A 与输入数据 WD 结合控制信号 size 向 RAM 中写入数据，并将写入地址和真正写入 RAM 的字输出。
3	复位	当 rst 信号为高电平时复位， RAM 所有字均清零。

电路图







10. 分支比较器

端口定义

端口名	位宽	方向	信号作用
A	32	Input	要比较的第一个数据 A
B	32	Input	要比较的第二个数据 B
CMPop	4	Input	比较种类选择信号
Out	1	Output	所选的比较条件是否为真

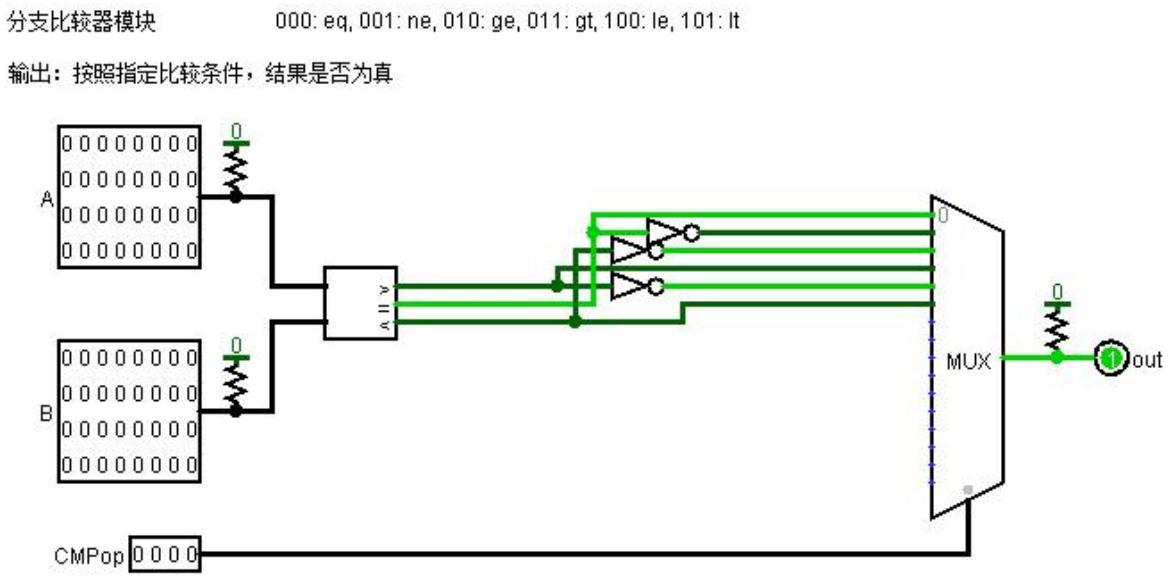
功能描述

用于分支类指令对数据的比较，当当前指令为分支指令并且此比较器输出为高电平，则判定分支条件为真，进行 PC 跳转。

选择信号与比较条件的对应表：

选择信号(十进制表示)	比较条件
0	A == B
1	A != B
2	A >= B
3	A > B
4	A <= B
5	A < B
其他	恒为假

电路图



二、控制单元与控制信号

控制信号真值表

控制单元根据指令的 `opcode` 与 `funct` 段的编码判断指令类型，并根据指令类型组合出控制信号真值表。由于本人在课下实现了 42 条指令，控制信号较多，故真值表以附件 excel 表格的方式呈现。

附件表格中包含的信息：

- 各个指令的 opcode 与 funct 编码
- 对指令种类进行的分类（分类标准一定程度上参考了 Mars 的源代码）
- 每条指令对应的控制信号真值，其中空白单元格表示，当前指令的执行不需要用到该路控制信号，在电路中设计为缺省值 0。
- 控制信号按 CPU 数据通路所属的阶段以及涉及到的元件进行分类。

电路设计

由于本人在课下添加了 42 条指令，指令数量和指令种类均较多，控制信号也较多，若采用 cscore 教程中的与或门阵列方式搭建，会导致电路非常庞大，走线密集且容易连错线，并且电路图不利于快速读懂（需要对照二进制表），所以本人采用了同样基于与或逻辑的另一种设计。

对于与逻辑，本人设计了一个内部基于异或门的对 opcode 或 funct 信号与给定 Constant 常量信号判断相等的模块，根据此模块输入的常量 opcode 以及 funct 可唯一地区分出指令种类。当需要扩展指令时，只需复制一份上述模块，并修改输入的常量信号以及输出的 tunnel 名称即可。

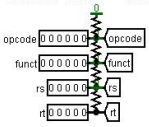
在判断指令种类与生成控制信号的或逻辑之间，本人引入了一层中间层的或逻辑，用于对指令进行分类，将指令分为（R型算术，R型移位，I型算术，内存读写，条件分支，跳转）六个大类。经过验证，这种分类方式较为直观，有助于控制信号的生成，简化控制信号的分类。

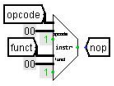
对于生成控制信号的或逻辑，则根据控制信号所属的阶段以及其影响的部件，分别根据真值表利用或逻辑进行建模，对于多位的选择信号（如 ALU, 比较器 等），采用 Priority Encoder 元件可快速建立且便于扩展。

电路图

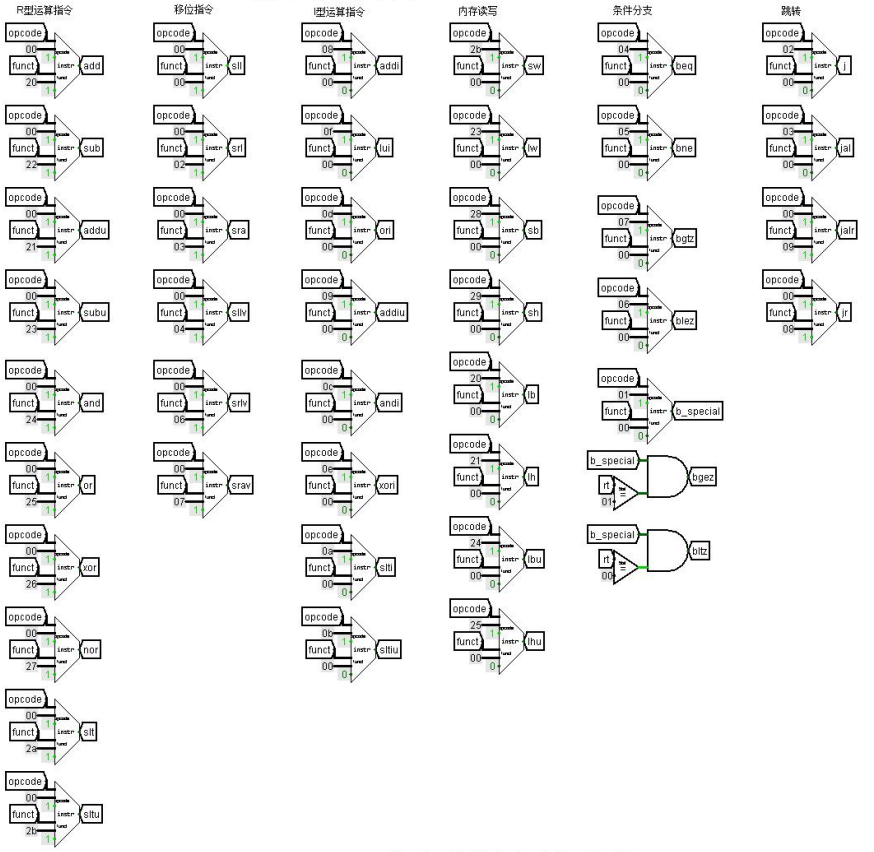
Control Unit: CPU的控制器单元

加指令方法：复制一个instr模块，修改opcode和funct的常量以及输出的标签即可

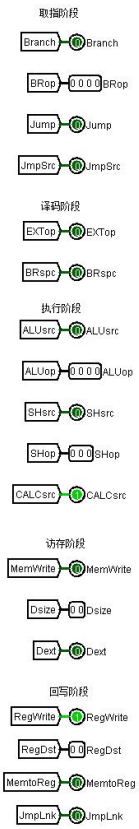




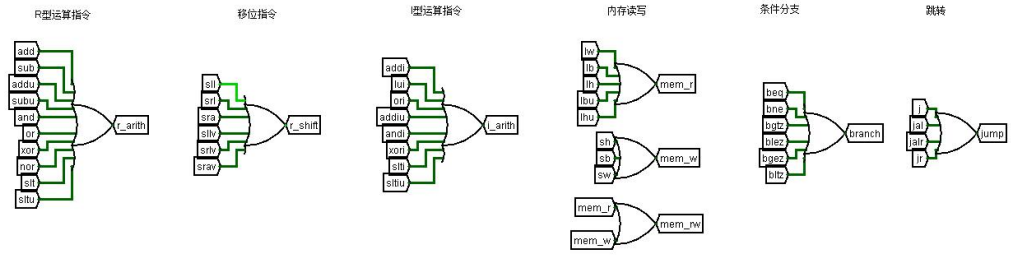
指令种类判断区



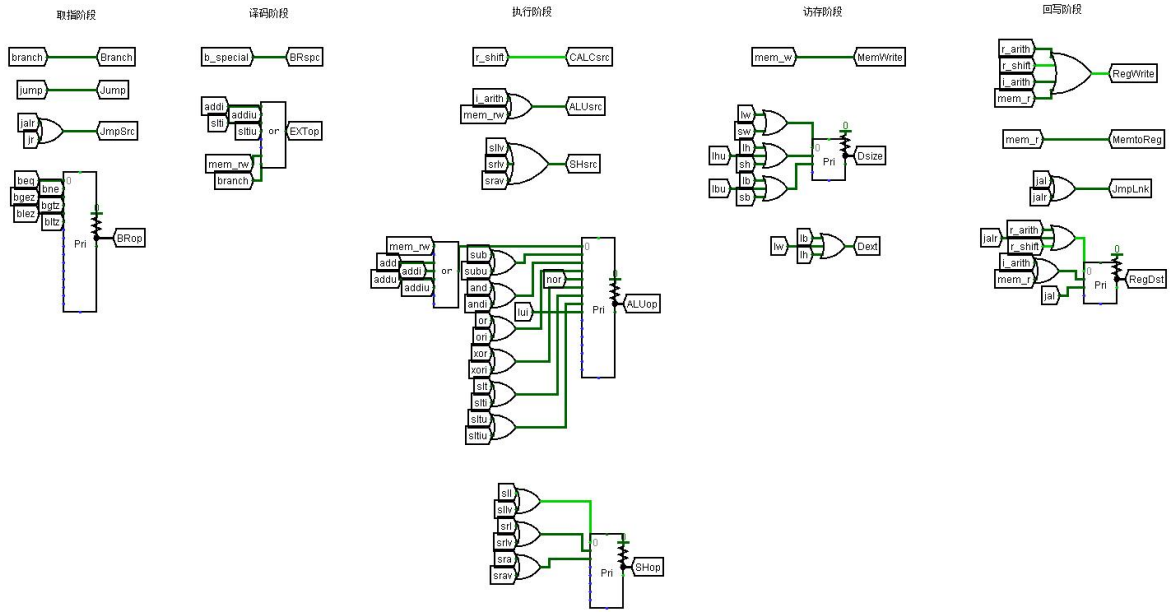
输出信号区



指令分类区(中间分类)



输出信号生成区



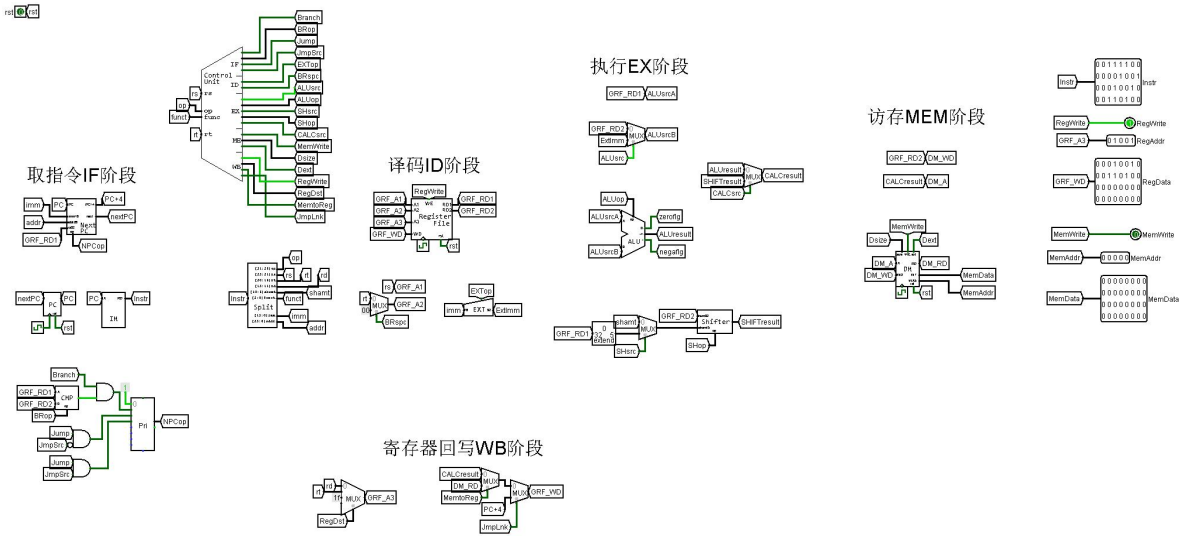
CPU 顶层模块 数据通路设计

顶层数据通路同样根据五个阶段来分别摆放进行设计，为简化布线，本人采取了利用 tunnel 将整个电路图分为若干分立但不失联系的组成部分，每个部件的输入采用 tunnel 进行命名。（尽管这种设计被某猪脚 diss 了，但这不一定不是一种直观且易于扩展的设计）

在设计过程中，每一条连线均被赋予了名称（即 tunnel 名称），且每条线均做到了有进有出。对于每个核心部件，如果有多种来源的输入，则对该输入信号加入 MUX 进行选择。在顶层模块中，每个部件位于上下边的端口一律为控制信号，左右边的端口一律为数据信号（地址也算作数据）。对于每个 tunnel 而言，若朝向右则表示该 tunnel 将信号传递给其他线，若朝向左则表示该 tunnel 接受一个来自左边的信号，tunnel 之间一定遵循自左向右的信号传递方向。每个部件的周围可放置若干多路选择器，用于选择该部件的**输入**（输出信号留给需要用到的后续部件进行选择）。

总体来说，设计过程中遵循了：每一路信号都有含义，每一路信号均做到有进有出，每一个部件的每一个输入都要清楚其来源。

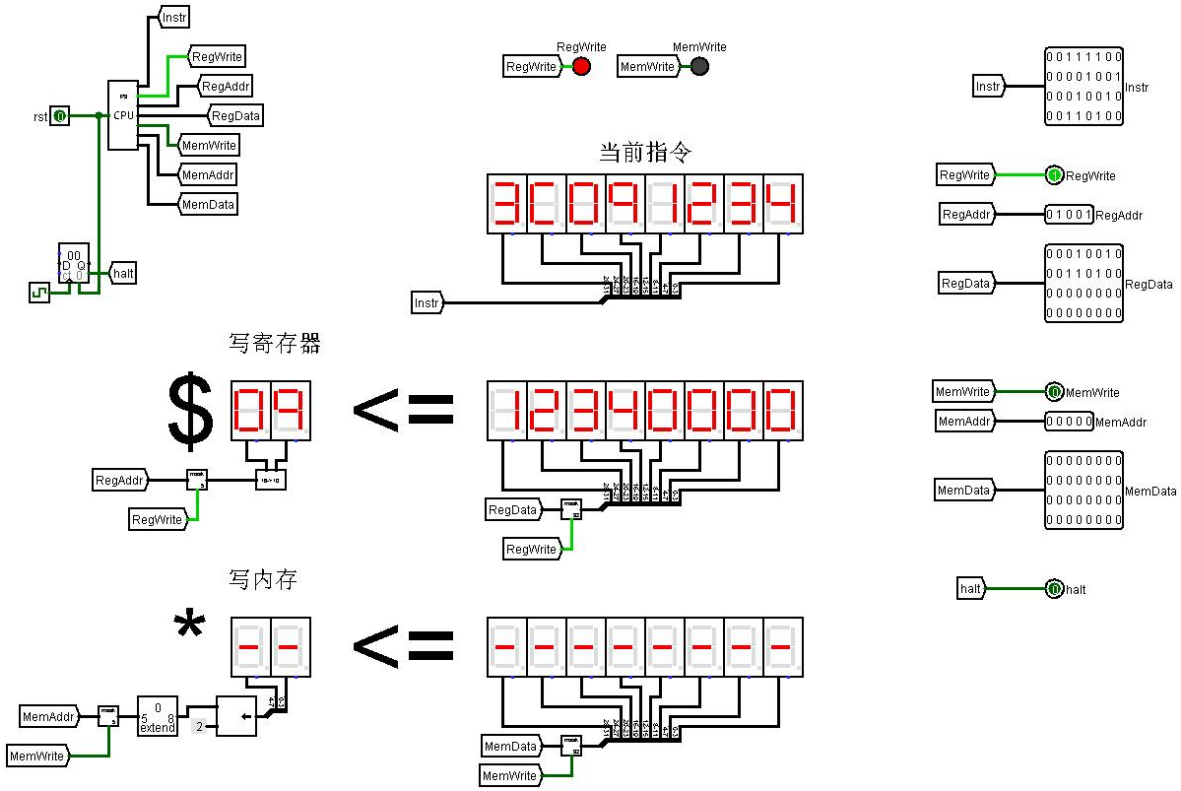
电路图



三、CPU 的测试

1. 测试电路

本人首先根据课程平台要求的端口将关键数据引出，并利用 Logisim 的数码管元件设计了一个非常直观的测试电路，实时输出每个周期的当前执行指令，即将写入的寄存器/内存的地址与数据。



2. 测试用例

测试用例的汇编代码部分为自己采用生成器生成，另一部分源自他人或者讨论区，此处仍然以附件形式放出。

3. 辅助工具

修改版 Mars

根据 Verilog 预习教程的挑战题中给出的 CPU 输出序列格式，将 Mars 的 jar 包解压后做出了一些修改，使得写入内存或者写入寄存器时按照指定格式输出信息，同时总结出了向 Mars 中添加课上助教自定义指令的方法，使得可以在课上用现场修改的 Mars 进行测试与验证。

输出处理与检查工具(即评测机)

共分为三部分，均采用 C 语言或者 C++ 语言编写。

1. 内存检查器：比对 Mars 导出的 .data 段内存与 CPU 运行汇编指令后的内存，输出出现差异的数据及其地址(如果有)，并给出评测结果
2. 输出序列检查器：比对修改版 Mars 的输出序列与 CPU 的输出序列，自动检查格式并跳过格式不符的行（无关行），比对 PC 值，写入寄存器或内存的差异，并给出评测结果。
3. Logisim RAM 导出数据整理器：将 Logisim 的 RAM 部件导出的长度不对齐并经过了一定压缩处理的数据整理成一个字一行，8位十六进制的输出，与 Mars 导出的格式统一，便于下一步调用内存检查器进行内存对比。

自动化测试

此部分包含一些批处理，主要有根据汇编代码生成十六进制机器码，利用 Mars 运行汇编程序并获取输出序列，调用评测程序比对标准输出，给出评测结果。

4. 测试方法

本人并未设计自动导入 ROM 并通过命令行调用 Logisim 的脚本，故仍然手动运行，通过测试电路的数码管输出与 Mars 给出的输出逐条比对进行测试，观察是否有不一致的输出结果。

四、思考题

1. 现在我们的模块中IM使用ROM， DM使用RAM， GRF使用Register，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

   答： GRF 使用 Register 以及 DM 使用 RAM 是合理的做法，在以后的 CPU 中也可以采用这种设计。对于 IM 使用 ROM ，在本阶段是合理的选择，但这种设计不一定适用于以后的 CPU 。因为真实的 CPU 需要反复加载汇编程序多次使用，更为合理的做法是采用可以多次读写的 RAM 存储器（FPGA 板子上自带）。并且真实的 CPU 中指令和数据全部位于内存当中，对于单周期必须将指令段和数据段分开，但以后的流水线 CPU 中可以将二者合并。

2. 事实上，实现nop空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

   答：空指令不进行任何操作，因此所有的控制信号一律输出 0 即可。

3. 上文提到，MARS不能导出PC与DM起始地址均为0的机器码。实际上，可以通过为DM增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

   答：由于在 Mars 中可以通过 CompactDataAtZero 参数，导出 DM 起始地址为 0 的机器码（并且这种机器码在本课程中采用最为普遍），因此 DM 默认初始地址为零即可。对于 PC ，则可以加入选择信号。本人采取的解决方案为，如果 PC 值大于等于 0x3000（ Mars 中 DM 地址为零时 IM 的起始地址）则自动将 PC 值减去起始地址 0x3000 ，这样可以做到无论是否遇到 j 指令，均可以正确地从 IM 中取出相应的指令字。

4. 除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式验证。 **形式验证**的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)”了解相关内容后，简要阐述相比于测试，形式验证的优劣之处。

   答：形式验证是根据某些规范或属性，采用数学方法证明某个程序设计或者电路设计的正确性。其优点在于可以较为完备地检测电路或者程序的设计是否正确。如果采用测试，无论构造了再多的测试数据，也只能说明程序在当前的测试集下是正确的，并不能确保程序或电路在任何情况下都是正确的（引用 accoding 的上机赛说明，Accepted != Correct ），而采用形式验证可以从数学上证明，某个设计是符合某种规范，实现了某种功能，是否满足某些属性，或者不存在某种漏洞等。形式验证的劣势在于可能数学上的证明比较复杂。