花开一季 叶落一地

一、Cache规格

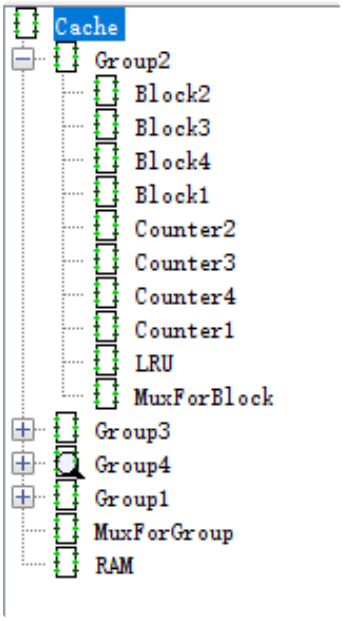
此电路要求模拟四路组相联的Cache，其中RAM大小为4K，Cache大小为256B，每一个Block为16B。Cache写机制为需要写入数据时既更新Cache中的内容，也向主存中写入。Cache的替换策略为LRU。
由既定的Cache规格可知，Cache中一组有4块，RAM中一组有64块，RAM的地址分为三块，从高位到低位分别为

组内块地址	组地址	块内地址
[11:6]	[5:4]	[3:2]

注：以下介绍Cache及其组成的图片和接口都是为了让同学们更好理解Cache的组成而举的例子，如果同学们有能力搭建更好的架构，在行为一致的前提下是可以随意搭建的。

二、模块介绍（整体）

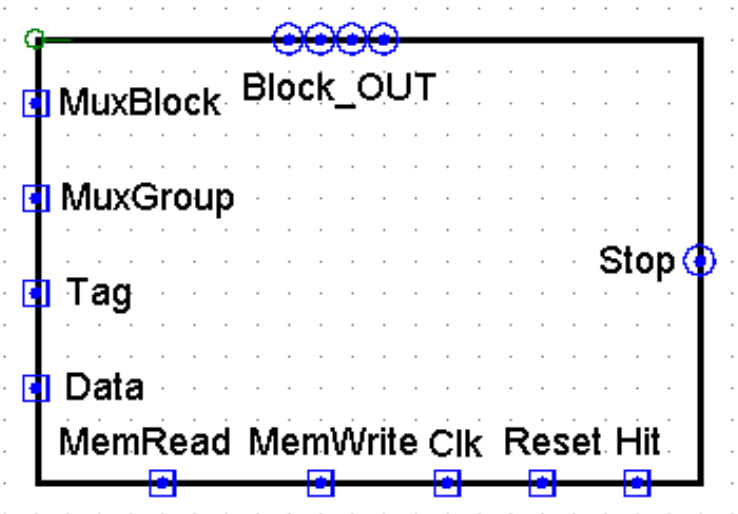
下图为标准的Cache模块总体的构成。



由上图并且联系我们Cache的规格可知，我们在Cache模块里用四个Group来表示Cache的四个组，在每个Group中用四个Block来表示每组的四个数据块。其中，Group中的Counter模块是为了计算此Block距离最近一次数据访问过了几个周期所搭建的，Group中的LRU模块是为了模拟LRU算法所搭建的模块，它可以通过每个Counter中的数据和其他信息来给出是否需要更新某个数据块。另外，为了能够使得在不改变我们的CPU内部走线的前提下将Cache放入CPU中，我们将RAM集成到Cache中。剩下的两个模块MuxForGroup和MuxForBlock本质上都是多选器。其中，MuxForGroup是为了选择从Cache Group传出的数据块而搭建的，MuxForBlock是为了选择从Cache Block传出的数据块而搭建的。
以上就是总体的模块介绍，推荐同学们自底向上搭建模块，可以先从最基本的模块比如说CacheBlock，LRU开始搭建。下面我们会从底层开始挨个介绍上图所使用的模块。

三、RAM

在搭建Cache时，我们会提供RAM模块，为了方便同学们更好的运用此模块，我们会在下方详细介绍此模块的接口和行为。如果同学们对RAM的某方面有疑问，可以去课程网站的讨论区或者点进RAM中自己查看即可。
下图为RAM模块的外观。

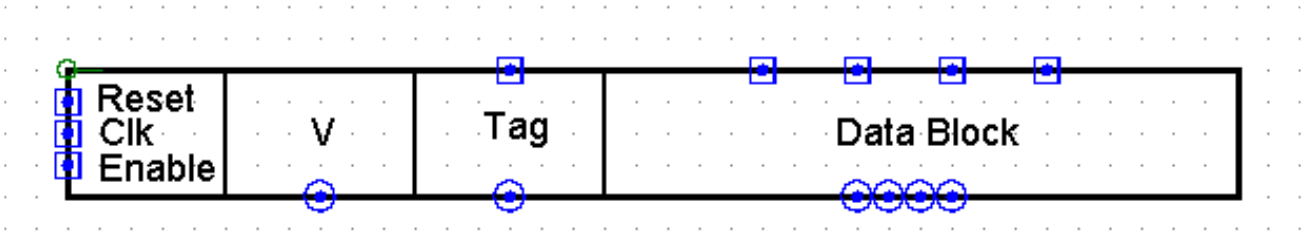


接口名称	IN/OUT	位宽	介绍
MuxBlock	IN	2	块内地址
MuxGroup	IN	2	组地址
Tag	IN	6	组内块地址
Data	IN	32	写入时传入的数据
MemWrite	IN	1	写入信号
MemRead	IN	1	读取信号
Clk	IN	1	时钟信号
Reset	IN	1	复位信号
Hit	IN	1	判断Cache是否命中的信号
上方的四个输出接口	OUT	共四个，每个接口32位，总共128位	输出要读取的Block内容
Stop	OUT	1	没命中时的暂停信号

先从输出开始说，RAM模块上方有四个输出接口，从左到右为高位到低位，即从左到右分别传出的数据位数为[127:96],[95:64],[63:32],[31:0]。包括其他模块的块数据传输所采用的四根32位的线，我们可以将它视作一根128位的线。这样，从左到右为高位到低位就可以很好理解了。
由既定的Cache规格可知，一个Block为16B，而Logisim软件中一根线的最大位宽为32位（4B），并且Cache和RAM是以块为单位进行数据交换的，所以在传输数据时就需要四根线的位宽。这样的话就引出了一个新的问题，Logisim软件内自带的RAM元件在一个周期内最多只支持32位的数据储存和输出，那么如何使用自带的RAM元件进行数据储存和输出呢?
为了解决这个问题，我们可以采用四个RAM并联在一起的结构，对于一个Block来说，每个RAM储存着此Block的1/4内容，那么这四个RAM就一起储存着所有的内容，我们只需储存数据时将数据存入特定的RAM中，读取数据一起读取即可。
我们搭建的RAM的主要功能就是存储数据和在读取数据且Cache不命中时进行需要的数据读取，那么就会使用到一个使能信号来决定是否读取RAM中的数据，当读取且不命中时，就需要读取RAM的内容。
如果Cache命中且不为写入状态的话，就不会访问RAM进行数据操作，这也是Cache真正省下来的时间所在。
为了让省下来的时间更加的直观，我们规定，当读取且没命中时，先暂停五个周期，然后第六个周期时再执行这条指令。也就是命中时执行指令需要一个周期，没命中时执行指令需要六个周期。为了方便同学们集中精力搭建Cache，我们已经为同学们将暂停信号提前搭建并集成进了RAM模块中，体现为Stop信号，需要注意的是，因为用的是米莉型状态机搭建的，所以使用Stop信号时需保持输入不变。
另外，为了支持我们的Cache写机制，我们提供的RAM在MemWrite信号置1时**会输出更新后的块数据**，也就是三个32位的未更改的数据和一个32位的即将写入的数据，这样可以使Cache在同一个周期完成更新操作，不用浪费更多的周期。

四、Cache Block

下图为Cache Block模块的外观。



接口名称	IN/OUT	位宽	介绍
Reset	IN	1	复位信号
Clk	IN	1	时钟信号
Enable	IN	1	使能信号
Tag上方接口	IN	6	Tag输入
Data Block上方接口	IN	4××32	数据块输入
V下方接口	OUT	1	有效位输出
Tag下方接口	OUT	6	Tag输出
Data Block下方接口	OUT	4××32	块数据输出

对于有效位，我们规定**在对应的数据块第一次写入时有效位变为1且除复位外以后不变**。

五、LRU

LRU，Counter，MuxForBlock，MuxForGroup这三个功能性模块，我们就只介绍他们的基本逻辑，接口介绍就省略不讲了。

LRU模块的逻辑通过此模块的名字就可以知晓大部分，就是通过每个Block对应的Counter和它们的命中情况来判断是否需要更新某个Block的数据，其中，Block对应的Counter计算的是此Block距离最近一次数据访问过了多少个周期，**计数器的数字越大，说明此Block距离最近一次数据访问过得时间越长**，更新时会找到数据最大的一个Block进行更新。在计数器都相等的情况下，我们规定数据更新的顺序为$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$。

下表为LRU的全部操作。

是否命中	是否写入	LRU模块操作
命中	写入	更新命中的模块
命中	不写入	不更新数据
未命中	写入	不更新数据
未命中	不写入	按照顺位更新计数器最大的Block

六、Counter

此模块是为了统计对应的Block距离最近一次数据访问过了多少个周期而搭建的。
我们规定，只有在**读取数据时才算作一次真正的访问**，所以**在写入时，不管有没有更新数据，都不算做一次访问。**
所以，计数器的逻辑就是只在读取时进行改变，即将更新对应的Block的数据时，自身归零，不更新数据时，如果计数值小于命中块的计数值 则该块的计数值加1，如果块的计数值大于命中块的计数值，则数值不变。最后将命中块的计数器清为 0。
计数器归零时，有三种情况，一是进行了复位操作，二是在读取时发现命中了，三是都没有命中即将按顺位更新对应的Block数据，这样就可以表明此Block距离最近一次数据访问过了多少个周期。其中，只有复位操作可以不在时钟上升沿进行。

七、MuxForGroup && MuxForBlock

此模块的逻辑是通过Cache是否命中和是否写入来选择某一个Cache Group或RAM的数据进行输出。
下表为MuxForGroup的全部操作。

组地址对应的组是否命中	是否写入	MuxForGroup模块操作
命中	写入	输出0
命中	读取	选择对应的Group数据进行输出
未命中	写入	输出0
未命中	读取	**先输出五个周期的0，第六周期输出正确的结果**

MuxForBlock

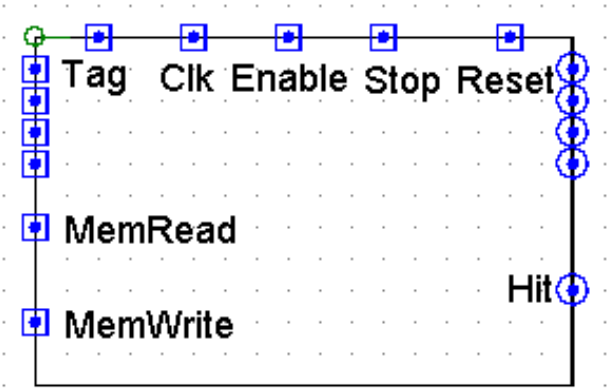
此模块的逻辑是通过Cache Group中每个Block是否命中来选择某一个Cache Block的数据进行输出。因为LRU模块的设置，所以不会发生多于一个Block命中的情况。
下表为MuxForBlock的全部操作。

是否有Block命中	MuxForGroup模块操作
命中	输出命中的Block的数据

介绍完MuxForBlock后，我们的底层模块介绍也就告一段落了，接下来我们重点介绍Cache Group和Cache这两个顶层模块。

八、Cache Group && Cache

下图为Cache Group模块的外观。

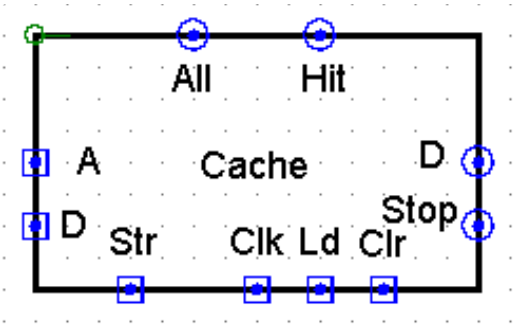


接口名称	IN/OUT	位宽	介绍
Tag	IN	6	输入Tag标志
Clk	IN	1	时钟信号
Reset	IN	1	复位信号
Enable	IN	1	使能信号
Stop	IN	1	读取没命中时的暂停信号
MemWrite	IN	1	写入信号
MemRead	IN	1	读取信号
左侧的四个输入接口	IN	4××32	将更新的数据传入的接口
Hit	OUT	1	命中信号
右侧的四个输出接口	OUT	4××32	将命中的数据传出的接口

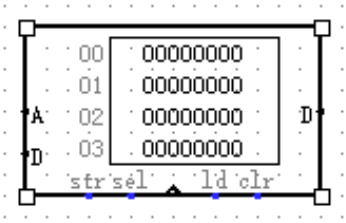
Cache Group内部总共有四个Cache Block，另外还有一个负责LRU算法的LRU模块，有一个负责选择输出的MuxForBlock模块和四个负责计数的Counter模块。

Cache

下图为Cache的外观。



下图为Logisim自带的RAM的外观



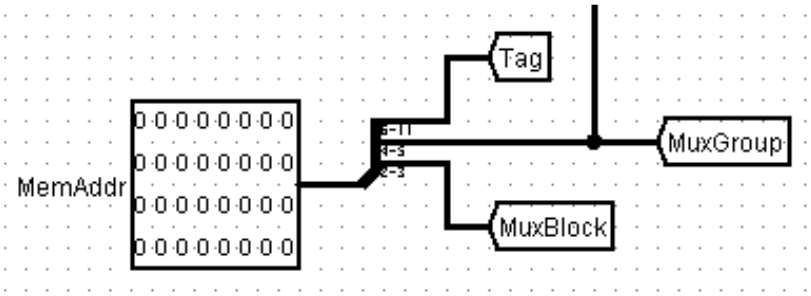
Cache的接口稍微要多一点，下面是多出来的接口介绍。

All和Hit接口是为了方便同学们了解自己Cache的效率而设置的，其中All接口是全部的访问次数，我们前面规定了，只有读取算作一次访问，所以这个接口也就是全部的读取次数，Hit接口是访问时的命中次数。同学们可以将所有的读取次数和命中次数做一个对比，和总周期数一起，体会一下Cache的作用。另外，这两个接口不是实时更新的，而是**延迟了一个周期，也就是如果当前周期命中，那么在下一周期这两个接口全部加一，在当前周期不变。**

$Stop$接口内部连接的是暂停信号，可以直接从RAM中引出来。

对比上面的两张图，Cache对于我们来说就像一个黑箱状态，它既能像RAM一样工作，在我们搭建的CPU中直接替换RAM，又可以在访问数据时相对于RAM节省大量时间。（当然在现实中Cache和RAM是分开的，在这个实验里为了方便同学们就集成起来了）如果同学们想像RAM一样看清里边的数据，调试的时候也可以引四条线出来，我们为了不破坏同学们已经写好的CPU结构，就不要求再做多余的结构了。

下图是地址的分线器。



就像我们一开始说的，六位Tag为组内块地址，两位MuxGroup为组地址，两位MuxBlock为块内地址。

在使用MuxForGroup后，筛选出来合适的块数据以后，我们还要用块内地址MuxBlock来进行最后的筛选。

最后选出来的就是我们要输出的32位数据。到这里，我们的Cache模块已经介绍完毕了，下面我们会再整体地梳理一遍数据进入Cache到底干了什么，走了哪条路。

九、整体梳理

读取数据时。

地址传入Cache前，经过分线器分出Tag位（组内块地址），MuxGroup位（组地址）和MuxBlock位（块内地址）。

其中MuxGroup位先被使用，由组地址来决定哪个组要被读取。

决定完毕后，Tag位被传入被选定的组，由组内的电路来判断是否命中，如果命中，数据就输入到MuxForGroup里进行下一步筛选，如果未命中，就发送信号（Hit）到RAM里，对RAM进行比较耗费时间的读取，RAM的数据也会输入到MuxForGroup里进行下一步筛选。

这时，数据都已经传输到MuxForGroup里了，下一步就是复用组地址进行筛选，筛选出来被选定的组传出来的块数据，再用块内地址筛选出来32位的数据进行输出。

写入数据时。

地址先被传入RAM中，再将RAM的整块数据传入Cache中，根据是否命中来进行Cache的数据更新。