**数据处理实验**

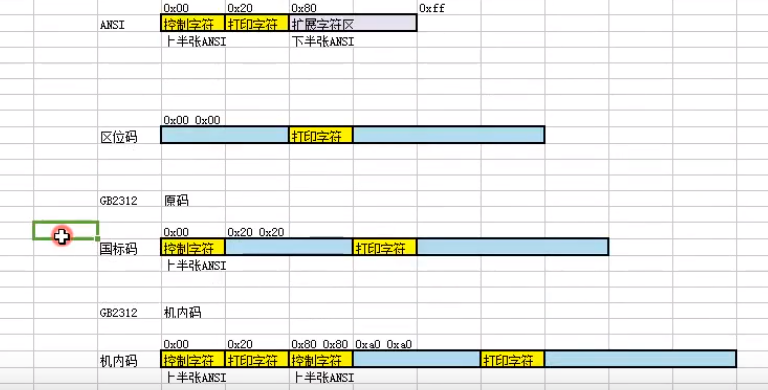
**汉字显示：**

**区位码，国标码，机内码的转换**

**国标在区位的基础上区字节和位字节都加上20H**

**机内在国标的基础上在加80H**

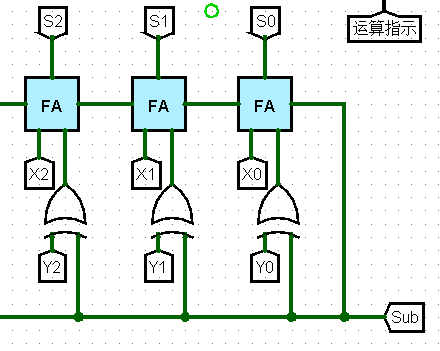
**所以从区位到机内直接加A0H**



**CRC校验有关的实验还没做**

**运算方法和运算器实验：**

**可控加减法器：利用FA，在每个被加数输入前进行异或调控是否取反，由Sub统一控制**

****

**先行进位电路：改进了串行FA时延太长的问题，把每个进位信号的函数抽象出两个函数：**

**进位产生函数和进位传递函数。根据函数的组成，经过两级门电路同步给出**

**各级的进位信号，打破了串行链。**

**快速加法器：加法器输出的结果函数为：Ci xor Xi xor Yi**

**将进位产生函数和进位传递函数用Xi和Yi表示输入先行进位电路**

**利用同步产生的Ci直接与Xi，Yi进行异或，同步得到最终结果，无串行链**

**在具体实验的时候，要将进位产生和传递函数每一项展开，利用多输入的门电**

**路，经过两级门电路将Xi和Yi调整为函数结果传入先行进位电路**

**总共经过5个逻辑门即可得到最终结果！**

**Xi和Yi转换进位产生函数和进位传递函数需要2级门电路**

**CLA74182内部需要2级门电路**

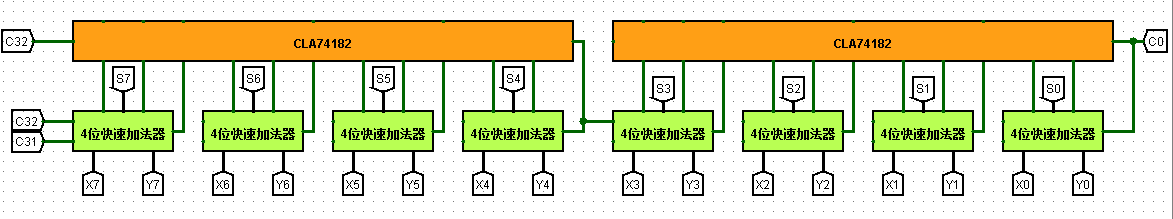
**最后Ci xor Xi xor Yi需要1级门电路**

**构建更多位数的快速加法器：**

**1.低级的快速加法器的G\*和P\*一定要封装在电路外部，方便作为输入传递给CLA74182**

**2.两个CLA74182之间进行连接的时候，注意进位位给入的位置(如图)**

**进位信号从4位快速加法器中给出相较从CLA74182中给出快了两个门电路的时延**

****

**阵列乘法器：将FA的进位信号斜向传递给下一个FA可以打破行间的串行链，但是最后需**

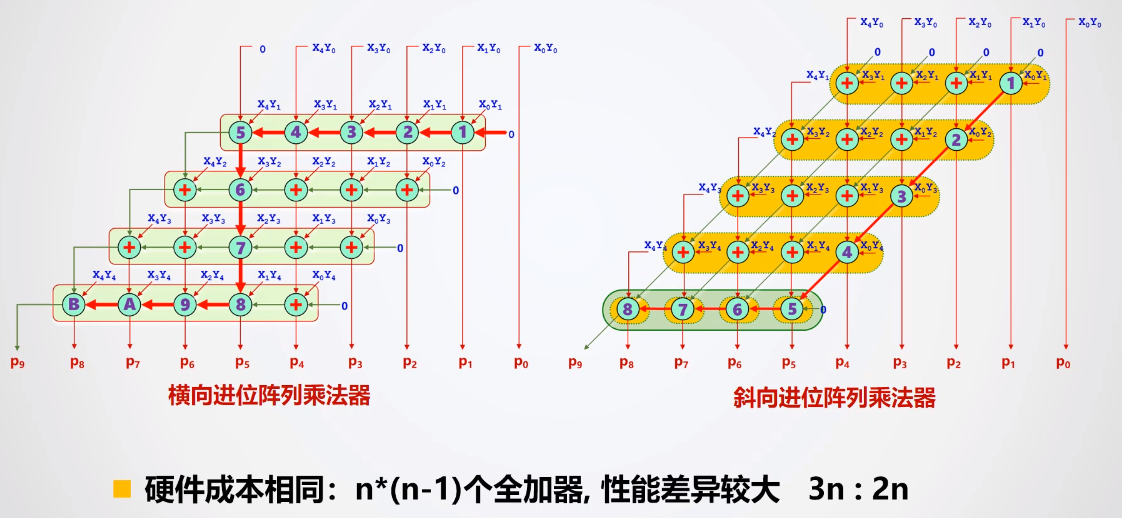
**要增加一行FA阵列用于调整进位，并且最后一行是串行的阵列！**

**分析一位乘法的结果：**

**1 \* 1 = 1 1 \* 0 = 0 0 \* 1 = 0 0 \* 0 = 0**

**跟与门的逻辑是一样的**

**分析两种阵列乘法器的成本和时延：**



**两者都需要N \* (N - 1)个全加器(部分可用半加器代替)**

**横向进位阵列乘法器需要经过(3n - 4)个全加器时延**

**斜向进位阵列乘法器需要经过2(n - 1)个全加器时延(最后一排串行)**

**因为全加器内部需要经过3级门电路，因此两阵列乘法器的门电路时延为：**

**横向：(3n - 4) \* 3T + T 斜向：(2n – 2) \* 3T + T**

**加T是用与门阵列并发，产生所有部分积所花费的门电路时延**

**分析：两位乘法时，两种方法时延和成本相同，位数越多，斜向进位越有优势**

**补码阵列乘法器：阵列运算电路操作的是纯数值，所有表示形式为有符号数的情况，符号位**

**单独运算，只有数值位参加阵列运算！**

**因此补码阵列乘法器相较阵列乘法器差别就在于，补码的符号单独运算**

**原码一位乘法：注意存储器件的时钟信号，使能信号，清零信号的统一，方便系统复位**

**注意移位的实现方法，计数器和比较器的搭配使用，二路选择器的使用**

**原码一位乘法的乘数，被乘数和结果都是原码形式，符号位单独运算**

**补码一位乘法：注意算数移位的实现方法**

**补码一位乘法的乘数，被乘数和结果都是补码形式，符号位参与数值位运算**

**32位ALU：已达到100**

1. **移位运算：**

**逻辑移位直接乘除就可以**

**算数右移先将补码转换成原码，符号为不动，原码在进行除法，有余数就在结果上加一，然后在转换成补码，记得之前的符号位要放在最高位，是有符号的除法**

1. **有符号的溢出检测注意加减法的区别，加法溢出只会发生在同符号相加，减法则是异号相减才会发生溢出**
2. **Logisim中的与或非等逻辑门组件都是按位操作的，有的需要调整初始设定，比如异或门**
3. **Logisim中的比较器是可以带符号也可以不带符号，因此具体使用的时候直接调整比较器的设置就行**

**五位无符号乘法流水线：**

**流水线 = 寄存器 + 组合逻辑 + 寄存器 + 组合逻辑 ……**

**每段流水线只做自己需要做的操作，跟普通乘法器对比(连续输入数据)：**

**乘法器：需要先做完一组乘法之后，再开始下一组乘法，两个结果之间间隔的是整个乘法过程的时间间隔。**

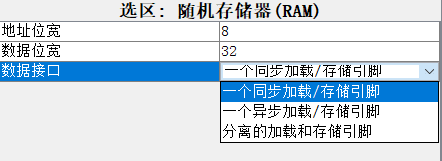
**流水线：一组乘数输入进第一段流水线，然后进入第二段流水线的时候，下一组数据就可以进入第一段流水线中，最后走过很多条流水线之后，得到第一组数据的结果，紧接着就是第二组结果，然后是第三组，他们相隔的时间是第一段流水线的时间间隔，而整个乘法过程中的某一个时间间隔肯定比整个乘法过程的时间小很多，因此广泛采用流水线方式设计乘法器**

1. **流水线的工作分的越详细，也就是细粒度越高，也就是分的段数越多，相邻数据的间隔就会越短，效率就越高。**
2. **流水线内部的组合逻辑是有时间延迟的(门电路的延迟)，如果频率太块导致时钟周期比门电路的延迟还小，那么就会产生错误，影响结果的正确性**

**存储系统设计实验：**

1. **了解RAM如何进行数据的输入：**

**Logisim中的RAM有三种形态**



**前两种的Data是双向线，控制下面的控制管脚决定读或者写(1读0写)，区别是有无时钟**

**最后一个是将读和写的Data分开了，一个在输入端一个在输出端**

**将数据放在Data端之后，为了能够连续输入，RAM的地址应该不断变化**

**因此需要在Address端接一个计数器，根据时钟的变化改变RAM的地址**

1. **字库实验**

**核心是RAM的容量扩展：**

**位扩展：位宽的扩展，将RAM中每个存储单元的位宽增加**

**字扩展：容量的扩展，将RAM的容量增加**

**这两个一定记清楚**

**位扩展只需要将地址值给入N个RAM，每个RAM中保存一部分数值，最后在相同的地址的作用下，所有的数值拼在一起就是最终的数值**

**字扩展时，地址总线的位数肯定是比RAM的地址线多(RAM的容量小，对应的地址位数少)，因此将地址总线的高位作为译码器的选择端，将译码的结果当做片选信号，最终的结果用多路选择器进行选择，选择端和译码器的选择端一样**

1. **MIPS RAM**

**MIPS Load/Store指令是唯一两种可以访存的指令**

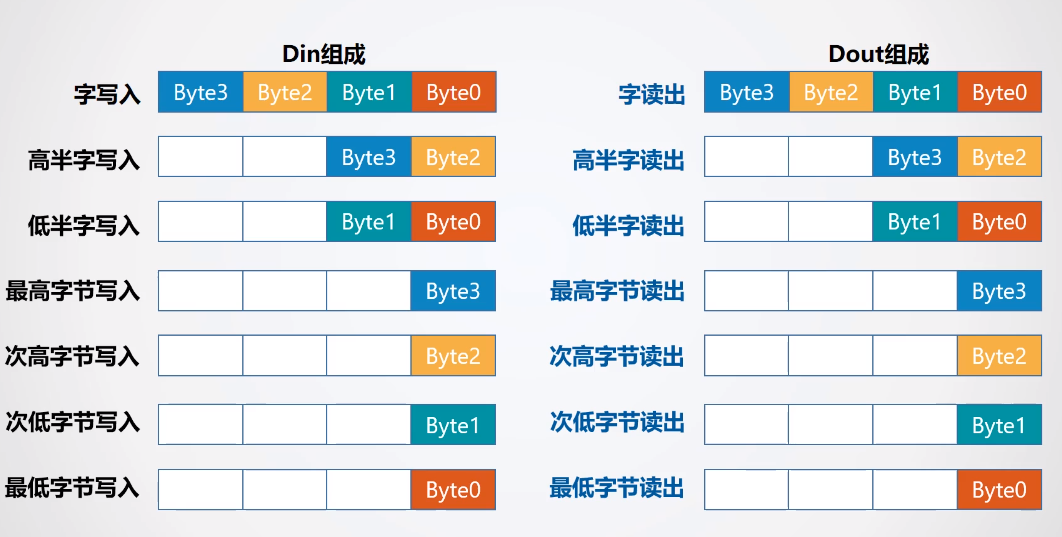
**访存的方式可以是字节访问、半字访问、字访问**

**具体的指令有lb,lh,lw,sb,sh,sw**

**其中l代表Load，s代表Store，b代表bite，h代表half，w代表word**

**访问主存的时候，半字访问和字访问需要进行强制对齐**

**半字访问忽略最低位，字访问忽略最低两位**



**整个实验主要分成四个板块：**

1. **地址强制对齐**

**强制对齐是根据不同的访存模式，将数据进行局部屏蔽(不屏蔽、最低一位或这最低两位)**

**然后在补上0，即实现了整数边界对齐的格式**

1. **输入数据的选择**

**不同的访存模式，RAM的数据输入可能会来自原32位数据的不同部分，因此在输入数据之前要根据模式进行数据的选择**

1. **片选信号的给出**

**写入、读出数据的时候要控制RAM的片选信号，而片选信号的控制取决于强制对齐后的地址的低两位和访存方式，两者通过纯组合逻辑就可以给出4个RAM的片选信号**

1. **输出数据的选择**

**不同的片选信号和地址低位都会导致输出数据的不同，但是要求有效数据显示在最低位，因此必须进行调整，也就是输出数据的选择**

**最终的成品在Logisim中已经保存，直接打开storage.circ即可**

1. **MIPS 寄存器组设计**

**MIPS寄存器组主要需要注意2点：**

1. **同时可以对两个寄存器进行读操作，因此，读操作对应的线路要复制一份**
2. **写操作时，存入的寄存器需要进行选择，同时，写使能信号也要进行选择，否则其他的寄存器会在一个寄存器被赋值的同时，由于写使能信号的开启被赋值为0**
3. **Cache映射机制与逻辑实现**

**Cache用的是SRAM，内存用的是DRAM因此，Cache要比内存访问快很多**

**并且在Cache miss之后进行数据的淘汰的时候，替换的是一整个数据块，而从内存中访问数据的时候是以字来访问的，因此，要多次访存才能拿到一个数据块，所以替换过程往往持续多个周期**

**如果是直接相连映射不需要做MISS之后的替换算法，也就是下图的第四点**



1. **这里注意技巧：先做一个模板，然后复制八份再去改标签**

**Cache槽的对应部分用位宽不同的寄存器来存放数据**

**Tag、Index、Offset从输入地址中分离**

1. **不同的地址映射方法对应的查找逻辑不尽相同**

**直接相连映射：直接将Index作为选择端从多路选择器中选取即可**

**全相连映射：将所有数据进行并发的比较**

**组相连映射：先用Index把数据进行分组，组内并发比较**

1. **命中 = 有效 + 标记相同**

**如果Valid为1并且Tag位于输入地址的Tag相同，那么就Hit**

**直接相连映射：将MUX选择的数据与标准数据进行比对，都相等就Hit**

**全相连映射：并发比较的结果都与标准数据进行比对，都相等就Hit**

**组相连映射：将选出的组内数据进行比对，最后结合Index的数据得出具体行的Hit**

1. **如果Valid和Tag有任意一个不满足要求(所有行都不Hit)，那么就MISS**
2. **如果是直接相连映射，那么在对应的行进行填充或替换数据**
3. **如果是全相连映射或者组相连映射，那么根据分组情况来找到对应的替换位置，根据替换算法进行替换，这里就要设计替换算法，因此直接相连映射是最简单的**
4. **数据块的载入在实验包里设计好了，主要就是根据准备信号来开启对应Cache槽的写使能就可以了，准备信号作为输入端的解复用器就可以做到**

**直接相连映射：Index作为选择端，准备信号作为解复用器的输入端即可**

**全相连映射：先用优先编码器查看有没有空行(Valid为0)**

**如果有，直接将数据送到解复用器的选择端**

**如果没有，启动淘汰算法，每行的频率计数器的值来进行淘汰**

**然后将数据送入解复用器选择端**

**组相连映射和全相连映射差不多，主要是有一个Index作为选择端的高位**

1. **MIPS CPU设计**
2. **单周期MIPS CPU设计**

**主要分为四个阶段：**

1. **连接数据通路**

**根据具体的指令的数据通路进行综合，照着课程上面的图画出来，很简单**

1. **设计指令译码逻辑**

**根据指令的op和fuct字段来判断具体是哪一种指令**

1. **设计操作信号产生逻辑**

**根据译码产生的具体的指令信号，产生不同的操作信号**

1. **设计ALU操作信号产生逻辑**

**ALU的操作信号不止一位并且每一条指令之间没有什么规律，所以相较于第三步，ALU的操作信号要单独设计**

**实验总结：**

**单周期的MIPS CPU 指令和数据存储器是分开的，使用32位宽的存储器时，正常访存的话，是字访问，但是BEQ和BNE指令中的分支地址都是字节访问，因此在访问指令存储器之前要先将指令逻辑右移两位，使字访问变为字节访问**

**在做排序的时候，因为指令是字节访问，所以每次当做地址的那个寄存器的值会加4，为了能够让数据存储器中的数据依次相邻存放，我们需要在LW和SW指令(访问数据存储器的指令)执行的时候，将于立即数相加的部分进行逻辑右移，原理同上，可以理解为，立即数给出了大致位置，寄存器中的值给出了间距，指令中的间距是按照字节来表示的，但是实际存储器是字访问，因此间距比实际大了4倍，所以，我们人为把间距缩小四倍(逻辑右移2位)**

**总结来说，指令中的偏移量都是按字节来的，如果在电路中存储器的访问方式不是字节访问，那么我们需要人为移位调整，如果不调整的话，对于指令访存来说会影响BEQ和BNE指令的跳转，对于数据访存来说会影响存放数据的位置**

1. **多周期MIPS CPU设计**

**相较于单周期主要的变化在于，不在采用哈佛结构，复用存储器将指令和数据一起存放节省成产成本，另外因为每条指令都不是一个周期内完成，因此，控制器不再是纯组合逻辑，需要设计时序，主要分为以下五步：**

1. **数据通路的连接**

**进行不同指令数据的综合即可**

1. **设计指令译码逻辑**

**同单周期MIPS CPU**

1. **设计微程序/硬布线控制器**

**微程序控制器：**

**先要设计不同指令的微指令并排序，根据排序的位置去设计指令入口转移逻辑**

**(当判断信号为1时，通过MUX选择，不选择下址字段，而是取出指令的入口地址)**

**设计微指令产生控制信号的时候一定要特别的细心，数据通路上每个信号都要考虑到**

**硬布线控制器：借用微程序控制器进行改造**

1. **设计操作信号产生逻辑**

**微程序控制器直接根据微指令产生控制信号**

**硬布线控制器根据不同的状态根据组合逻辑产生**

1. **设计ALU信号产生逻辑**

**微程序控制器在微指令中单独设计2位ALU控制信号：**

**00表示加法，01表示减法，10表示根据fuct字段进行变化**

**硬布线控制器控制组合逻辑调控出跟微指令一样的入口地址即可**

**实验总结：**

**同单周期，指令的字节访问需要我们人工进行移位**

**设计微指令产生控制信号的时候一定要细心，实验的时候运行不出排序结果，检查了一下午才查出来问题在于分支指令跳转的时候ALU的控制信号选错了！**

**值得提出的一点是：**

**在微程序控制器中，LW和SW指令除了取指令和去操作数外，实际执行过程中的第一步也是相同的，因此可以进行合并，这就需要我们在一个已经设计好的第三条指令之后，进行分支跳转，用原来的微指令格式肯定是无法实现分支的，因此，把原先的判断字段p 从一位改为两位，指令来源的选择就从2种变成了4种，那么当p 字段为10或者11的时候，我们可以去通过三态门的选择给出不同的指令来实现跳转。**

1. **24条指令单周期MIPS**

**在8指令单周期MIPS的基础上加入J型指令，syscall的显示功能…**

**在原有的基础上扩充数据通路，进一步的综合，设计方法差不多，最后通过benchmark测试**