

存储系统综合实验

实验内容

1. 存储扩展实验

1. MIPS寄存器文件设计



实验一 存储扩展实验

实验目的： 掌握存储扩展基本原理

实验内容： 设计字库文件， 利用指定规格存储器进行存储器字扩展和位扩展， 具体见实验文档。

实验要求： 利用给定规格的ROM部件（2个4K*16位ROM， 3个4K*32位ROM， 7个16K*32位ROM）构建GB2312 16*16点阵字库存储器电路。电路输入为汉字区号和位号， 输出为表示对应汉字16*16位点阵信息的256位数据。

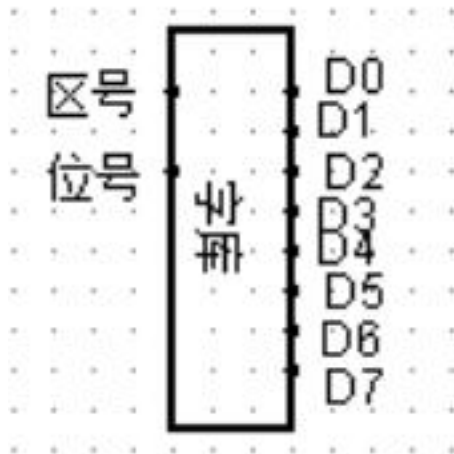


预备知识 - 存储器扩展

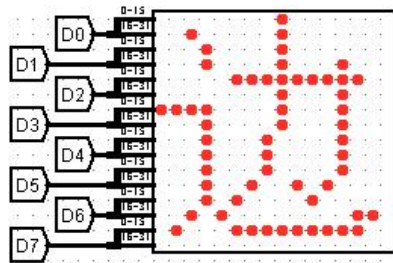
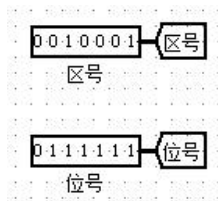
- 由于现有存储器的容量和字长与目标存储器的容量和字长可能存在差异，一般采用位扩展、字扩展、字位同时扩展等方法来组织现有存储器，从而实现与目标存储器一致的功能。
- 具体原理请自行阅读教材 P132



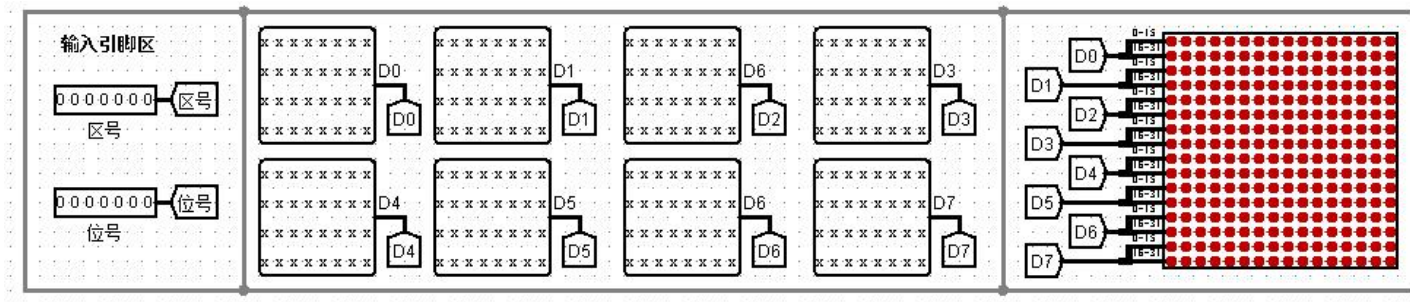
字库电路的封装形式



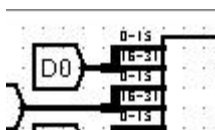
- storage.circ中的字库电路即时我们要完成的字库电路
- 输入：汉字在GB2312编码中的区号和位号。
 - 注意：这里区号引脚和位号引脚都只接受7个位数据，这是因为GB2312-80将汉字分为94个区和94个位，故7个位（可表示0-127）即可涵盖区间[0,94]。以“边”字为例，其GB2312编码为1763，17表示区号，63表示位号，故为了显示“边”，我们需要往区号引脚输入0010001（17），往位号引脚输入0111111（63）。
- 输出：GB2312区位码对应的汉字字形码。
 - 电路总共有8个输出引脚D0-D7，每个引脚输出32位数据，总计输出 $8 \times 32 = 256$ 位数据，从而可以表示 16×16 字形点阵所需要的256位信息。



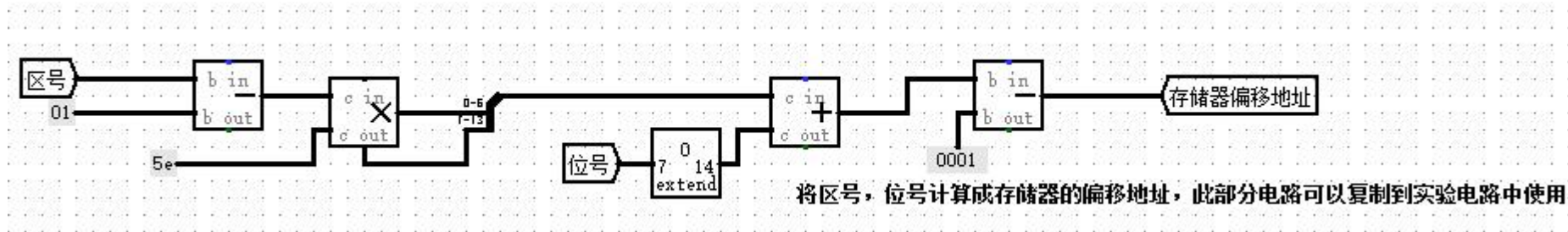
字库电路输入输出引脚



- 左：输入引脚区，通过 区号tunnel 和 位号tunnel 可以分别获得表示区号/位号的7个位
- 中：8个PIN组件分别连接至字库电路的8个输出引脚，用以观察字库电路输出的8*32位字形码
- 右：字库电路的8个输出引脚（D0-D7）连接至16*16 LED点阵屏上，从而显示汉字的形状。
 - **注意：**16*16汉字点阵每一行需要2个字节来存储16位信息，直观做法是给电路设置16个输出引脚，每个引脚输出16位信息，即每个引脚用以显示一行。而实验给定电路使用8个输出引脚，每个引脚输出32位信息，即每个引脚用以显示两行。以输出引脚D0为例，其首先经过一个splitter切分成两条线路（一条包含位0-15，一条包含位16-31），之后再连接至LED点阵屏上，用以显示两行。



参考电路实现 - 通过区位码获得存储器偏移地址

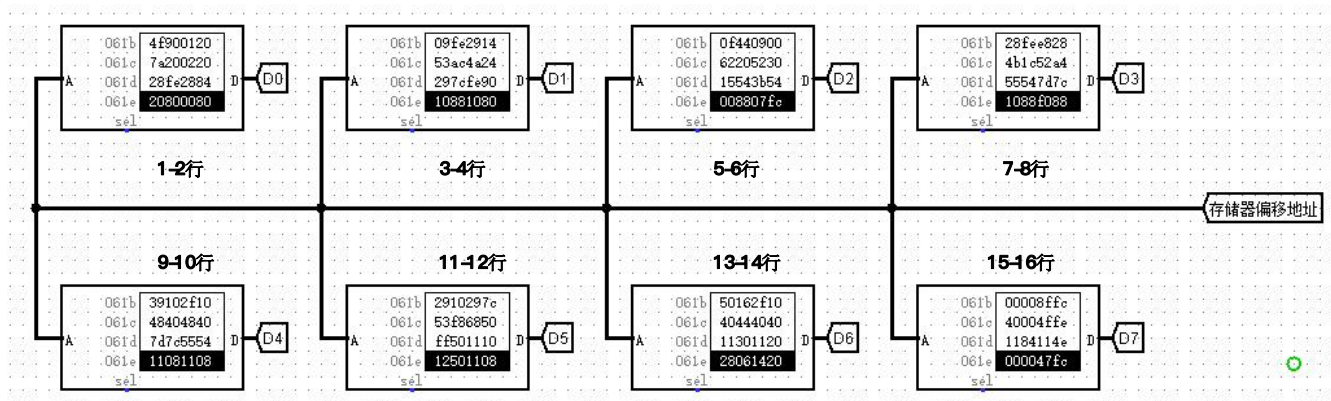


问题：如何通过GB2312区位码获得对应汉字的16*16点阵码？

- 事先在存储器中存储点阵码
- 对区号和位号经过一系列处理之后计算出存储器的偏移地址，每个偏移地址对应一个16*16汉字点阵码

注意：这部分电路可以直接复制粘贴（在参考电路使用箭头选定后复制，在目标电路中CTRL-V）到我们要实现的字库电路中，不用深究

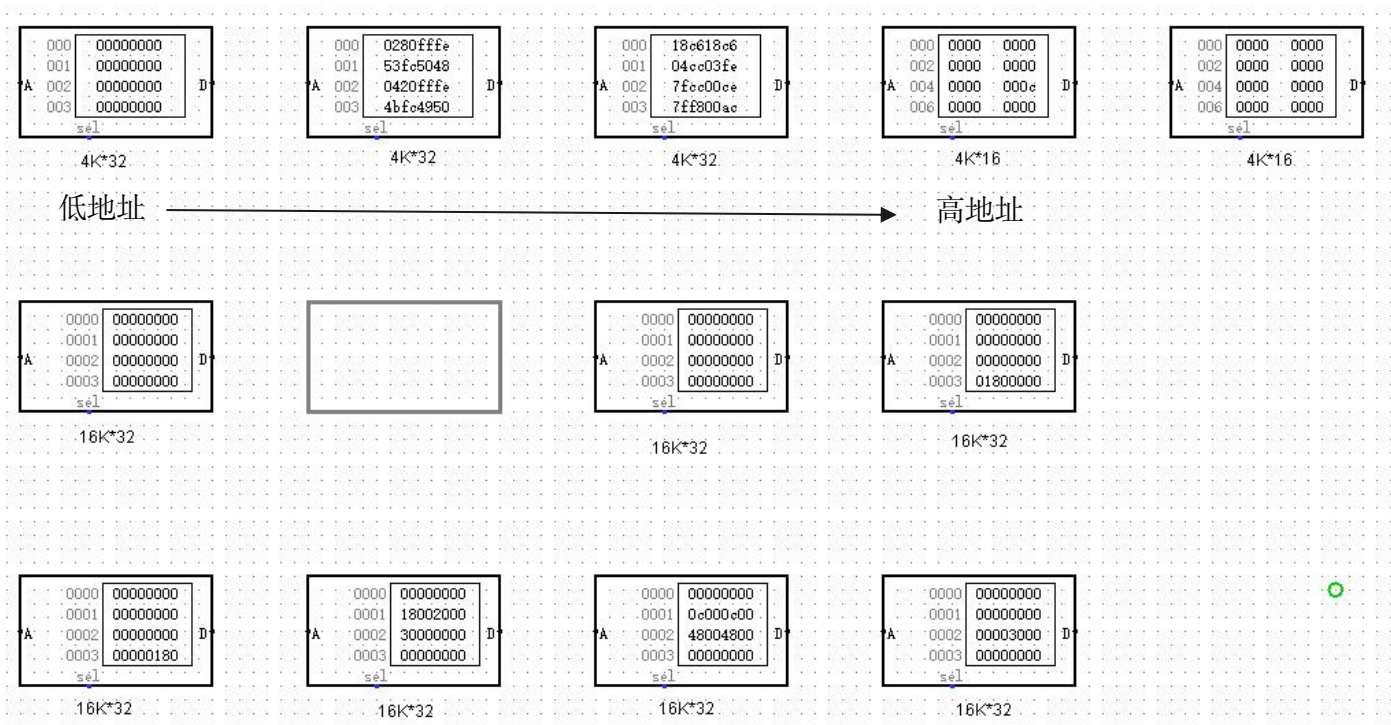
参考电路实现 - 一次获得表示汉字的256位点阵码



问题：我们现在只有8个16K*32位ROM，对单个32位ROM寻址一次只能获得4个字节数据（32位），如何才能一次获得在16*16点阵屏上表示单个汉字所需的256位点阵码呢？

- 位扩展：既然一个32位ROM寻址一次可以获得32位数据，那么我们可以**每次使用同一个存储器偏移地址同时对8个32位ROM进行寻址**，这样单次寻址就可以获得8*32=256位数据了，其中每个32位ROM保存16*16点阵码的**固定两行**（表示一行需要16位，两行即32位）
- **注意**：每个只读存储器中事先已经保存每个汉字对应行的点阵码，不要擅自改动。

待实现字库电路



左图为待实现的字库电路:

组件:

2个4K*16位ROM

3个4K*32位ROM

7个16K*32位ROM

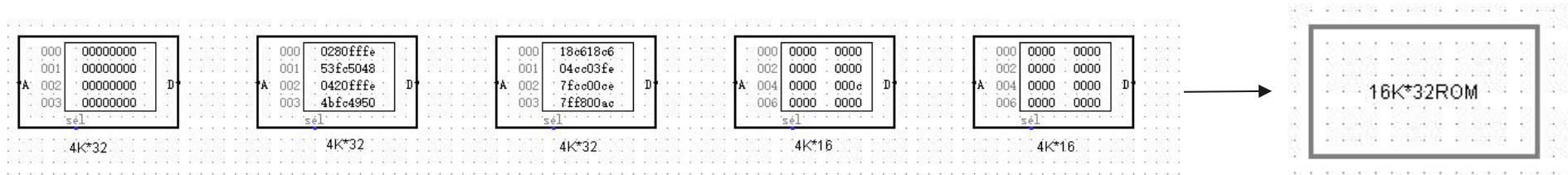
(注意: 对于电路图
中的3个4K*32位
ROM和2个4K*16位
ROM, 由于数据已
经导入到ROM中,
所以顺序固定为从左
到右地址增加, 即在
进行字扩展时, 注意
地址区间的划分)

问题

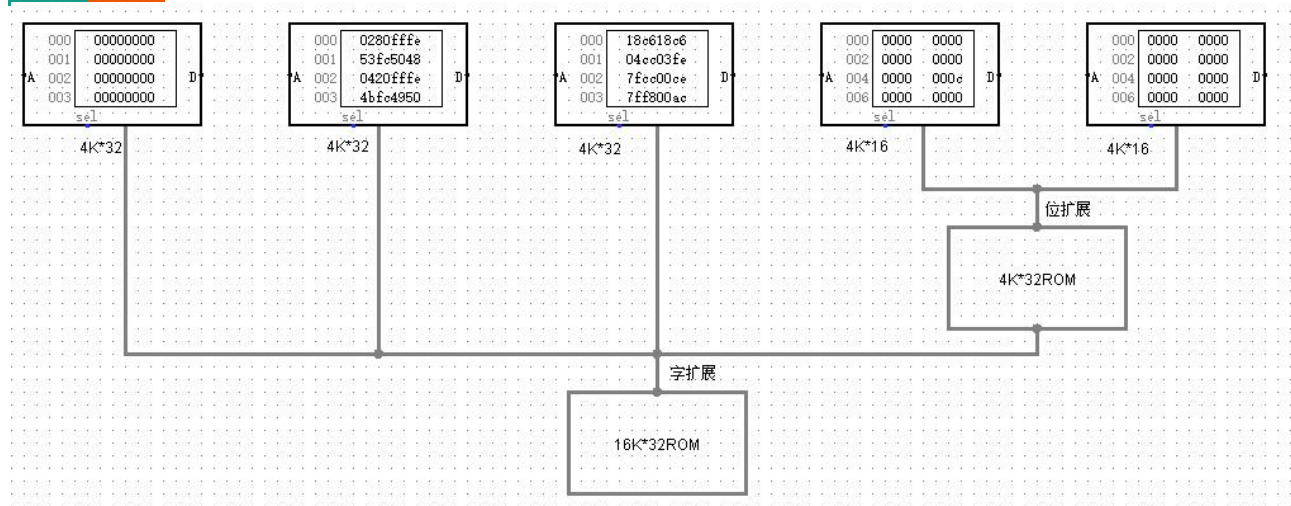


参考字库电路使用8个16K*32位ROM进行位扩展，实现了一次获取16*16点阵屏需要的256位点阵码，现在要求使用2个4K*16位ROM，3个4K*32位ROM，7个16K*32位ROM来实现相同功能的电路，该如何做呢？

- 与参考电路相比，我们缺少一个16K*32位ROM，多了2个4K*16位ROM和3个4K*32位ROM
- 利用字扩展（容量扩展，地址线扩展）和位扩展，将现有的2个4K*16位ROM和3个4K*32位ROM组织成一个16K*32位ROM




存储器扩展具体思路

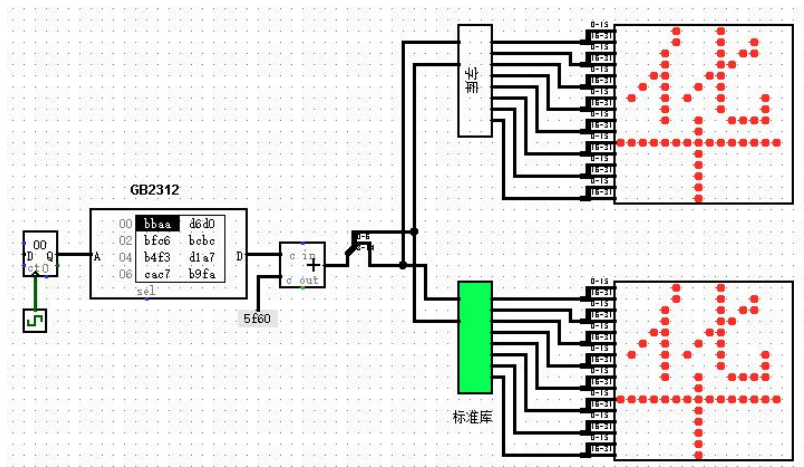


- 利用**位扩展**，将现有的2个4K*16位ROM组织成1个4K*32位ROM
- 利用**字扩展**，将现有的3个4K*32位ROM和上一步得到的4K*32位ROM组织成一个16K*32位ROM
 - 注意：由于数据已经导入到ROM中，所以顺序固定为从左到右地址增加，因此在字扩展过程中，需要注意电路组织方式。
 - 提示：使用splitter和multiplexer

测试字库电路



- storage.circ中的“字库测试”电路用于测试我们实现的字库电路
- 测试方法:
 - 不断点击  触发时钟信号，或者按下CTRL+K使电路自动仿真，对比字库电路和标准库电路输出是否一致
 - 字库电路输出的汉字和标准库输出的汉字一致，则表明电路正确，反之，则表明电路不正确
- 右图注释
 - 由实验一我们知道，区位码+A0A0=机内码，机内码是计算机内存储和处理汉字时使用的编码，参考教材P44
 - **GB2312存储器存储汉字机内码**（区位码+A0A0）
 - 机内码首先与5F60（A0A0的补码）相加，**其目的是利用补码加法实现减法，从而获得对应区位码**（参考实验二用补码加法实现减法）。
 - 从区位码提取出区号和位号，输入到字库点库
 - 字库电路输出点阵码，使得LED屏显示汉字



实验二 MIPS寄存器文件设计

实验目的：为MIPS CPU构造核心功能部件，进一步熟悉多路选择器，译码器，解复用器等Logisim部件的使用

实验内容：参考实验文档

实验要求：

- 利用logisim平台构建一个MIPS寄存器组，内部包含32个32位寄存器（实际4个）
- 为减少实验中画图工作量，实验工程文件中对5位寄存器地址进行了简化，具体见引脚示意图，**最终只需实现4个寄存器（0-3号），0号寄存器恒为0**
- 注意时钟信号和电平信号不要混连，时钟仅仅触发状态改变

MIPS寄存器文件的封装形式

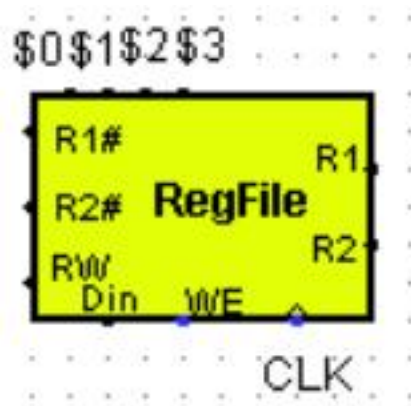


输入:

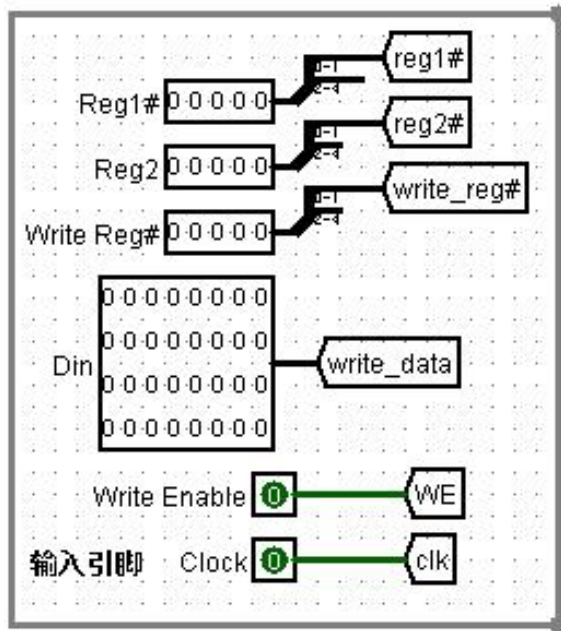
- R1#: 5位 读寄存器1编号
- R2#: 5位 读寄存器2编号
- RW: 5位 写入寄存器编号
- Din: 32位 写入数据
- WE: 1位 写入使能信号, 为1时, CLK上跳沿将Din数据写入RW指定的寄存器中
- CLK: 时钟信号

输出:

- R1: 编号为R1#的寄存器的值
- R2: 编号为R2#的寄存器的值
- 观察引脚: 下面四个引脚只用来观察寄存器当前的值
 - \$0: 0号寄存器的值, 写入无效, 恒为0
 - \$1: 1号寄存器的值
 - \$2: 2号寄存器的值
 - \$3: 3号寄存器的值



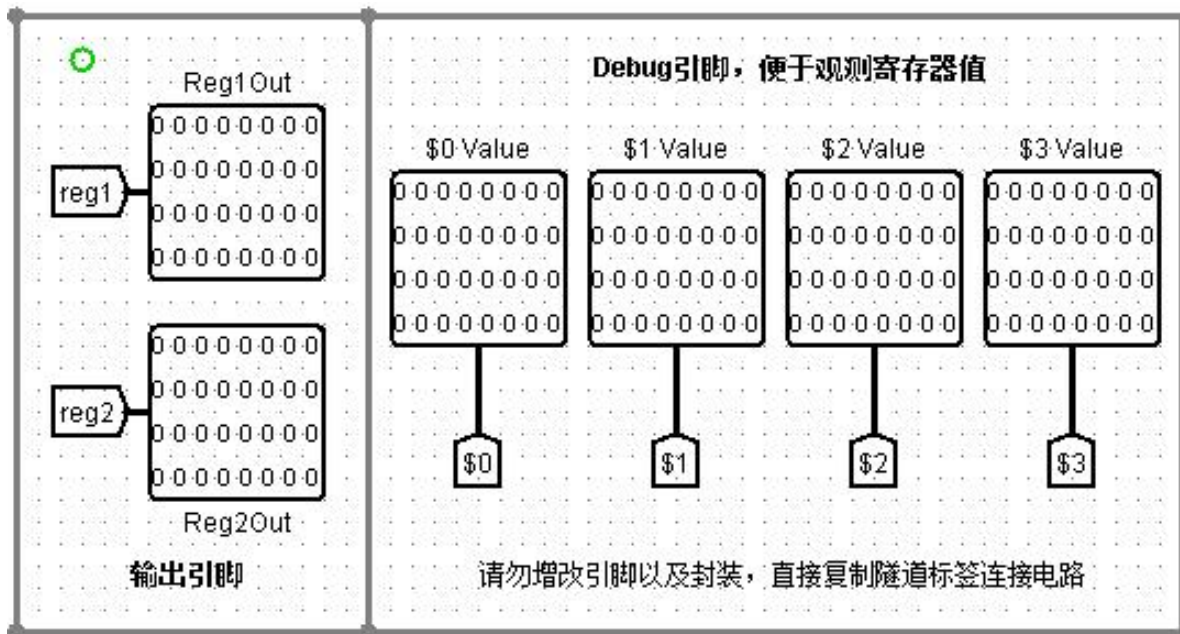
MIPS寄存器文件电路的输入引脚



- 分别通过reg1# reg2# write_reg#获得输入的三个寄存器编号，因为实验只要求实现0-3号寄存器，所以只需提取5位编号输入的低2位（00/01/10/11）
- 通过 write_data 获取要写入寄存器的32位数据
- 通过 WE 获取写入标志
- 通过 clk 获取时钟信号

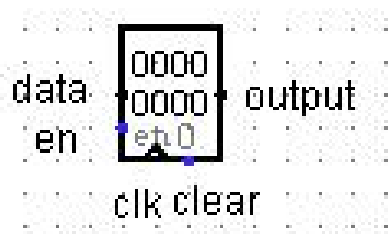
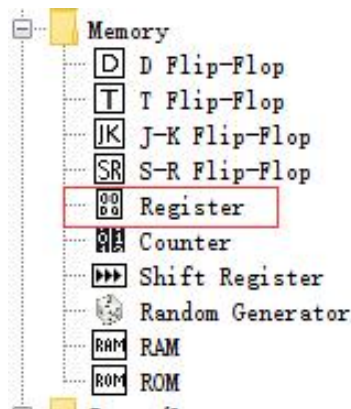


MIPS寄存器文件电路的输出引脚



- 将读寄存器1的值输出到 **reg1**
- 将读寄存器2的值输出到 **reg2**
- 将4个寄存器当前保存的值输出到 **\$0** **\$1** **\$2** **\$3**

MIPS寄存器文件电路实现 – 寄存器组件介绍



- Memory目录里有Register组件，实验需要用到4个
- 寄存器输入引脚：
 - data: 可能会写入到寄存器的值
 - en: 当en=1时，clk信号会使得寄存器保存当前data引脚的值；当en=0时，clk信号不会使寄存器保存当前data引脚的值
 - clk: 时钟信号
 - clear: 当clear=1时，将当前寄存器值异步置零（清零功能不做要求，感兴趣自行实现）
- 寄存器输出引脚：
 - output: 输出当前寄存器保存的值

MIPS寄存器文件电路实现 – 寄存器选择输出

问题：MIPS寄存器文件电路内部有编号0-3的4个寄存器，如何才能实现只将编号为reg1#和reg2#的寄存器的值输出到reg1和reg2输出引脚呢？

- 利用多路复用器（multiplexer），多路复用器可以通过select bits选择多个输入中的一个进行输出
- 因此，我们可以将四个寄存器的输出引脚同时接到多路复用器的输入引脚上，将reg1#和reg2#作为多路复用器的select bits，从而选择指定输入进行输出。

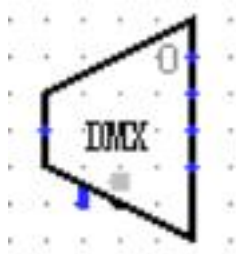


MIPS寄存器文件电路实现 – 寄存器选择写入



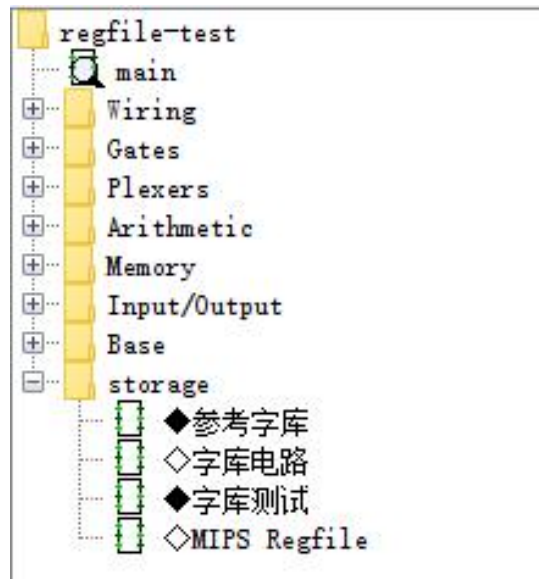
问题：MIPS寄存器文件电路内部有编号0-3的4个寄存器，如何才能实现只将Din引脚输入的数据写入到编号为write_reg#的寄存器中呢？

- 利用解复用器（demultiplexer），解复用器可以将输入只通过select bits指定的线路输出。
- 因此，我们可以将write_data同时连接到四个寄存器的data引脚（由于0号寄存器恒为0，因此可以忽略data引脚，或者将en置0），将we引脚作为解复用器的输入，将write_reg#作为解复用器的select bits，通过解复用器将要写入数据的寄存器的enable引脚置1，下个时钟信号到来时，只有该写入寄存器的enable引脚为1，因此write_data数据只会写入到该寄存器中（其余寄存器enable引脚为0，时钟信号不会触发寄存器写入），从而实现寄存器选择写入。



MIPS寄存器文件电路测试

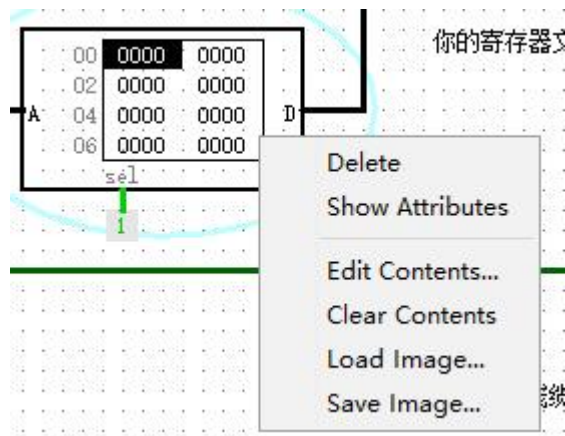
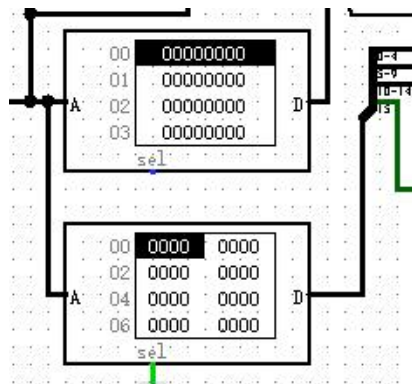
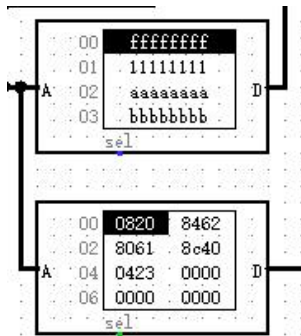
- 电路完成后，打开文件夹内的regfile-test.circ文件
 - main电路就是测试电路
 - storage目录即使我们之前的工作目录，里面有我们完成的待测试MIPS Regfile电路



MIPS寄存器文件电路测试



- 由于实验没有自带测试数据，因此两个存储器内容都为0。
- 为了进行测试，这里手动给出几组测试数据以及正确视图，
同学们需要将regfile-test-data目录下的data数据集（data_set）
和指令数据集（instruction_set）分别手动导入到两个存储器
中，也可以通过鼠标右键点击存储器写入自己的测试数据。
- 若存储器如下图所示，表明成功导入测试数据集
- 触发时钟信号不断进行测试



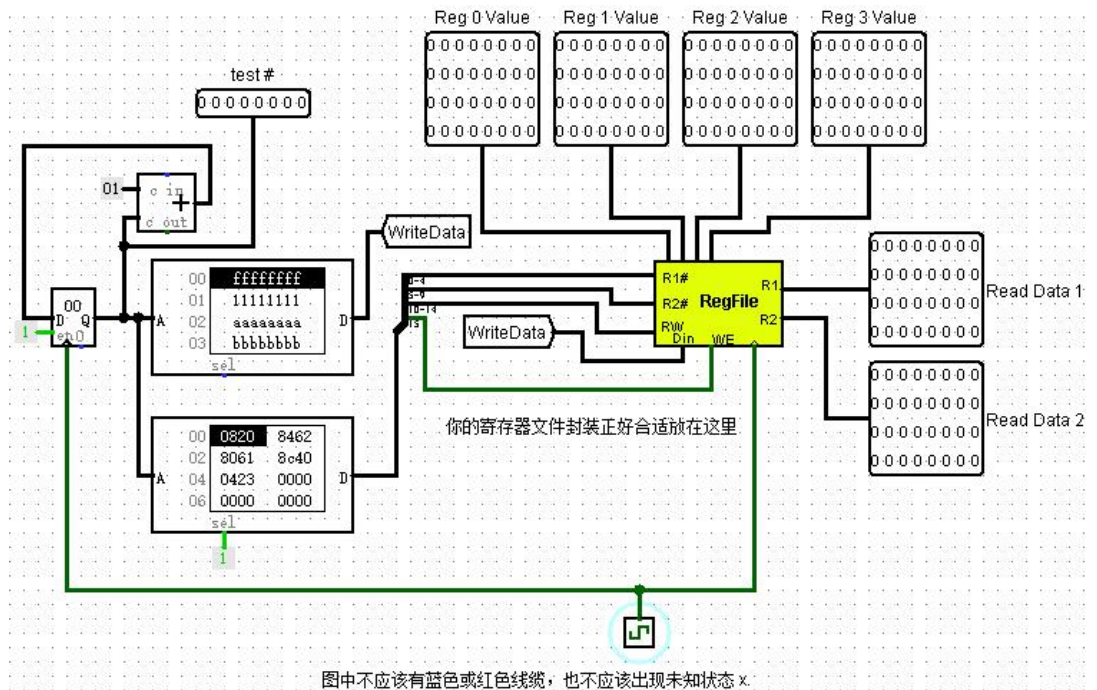
MIPS寄存器文件电路测试数据及正确视图

0. 执行指令前

时钟信号有两个作用:

- 增加存储器偏移地址，从而读取下组数据
- 若当前指令中WE=1，将Din引脚的数据写入到编号为RW的寄存器中

注意：时钟信号不直接影响R1和R2，R1和R2始终为当前存储器输出指令指定的编号为R1#和R2#的寄存器值



MIPS寄存器文件电路测试数据及正确视图

1. 读0号和1号寄存器

$$R1\# = 0$$
$$R2\# = 1$$
$$RW = 2$$

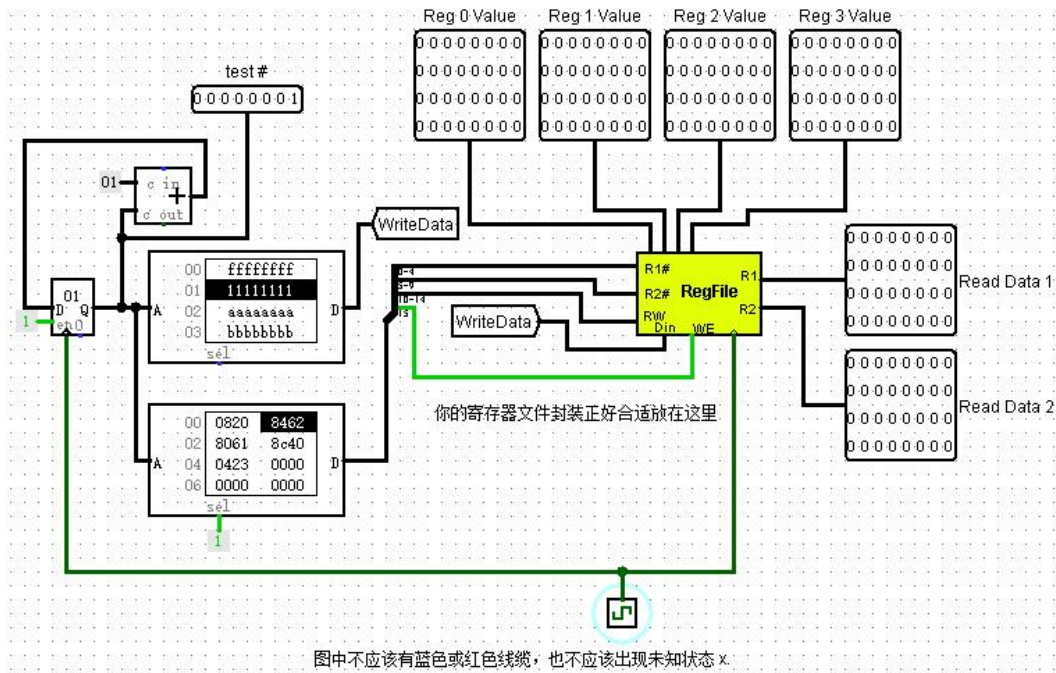
Din = FFFFFFFF

$$WE = 0$$

对应数据存储器值: FFFFFFFF

对应指令存储器值: 0820

执行完该条指令之后，结果如右图所示



图中不应该有蓝色或红色线缆，也不应该出现未知状态 x。

MIPS寄存器文件电路测试数据及正确视图

3. 读1号寄存器和3号寄存器，写0号寄存器

R1# = 1

R2# = 3

RW = 0

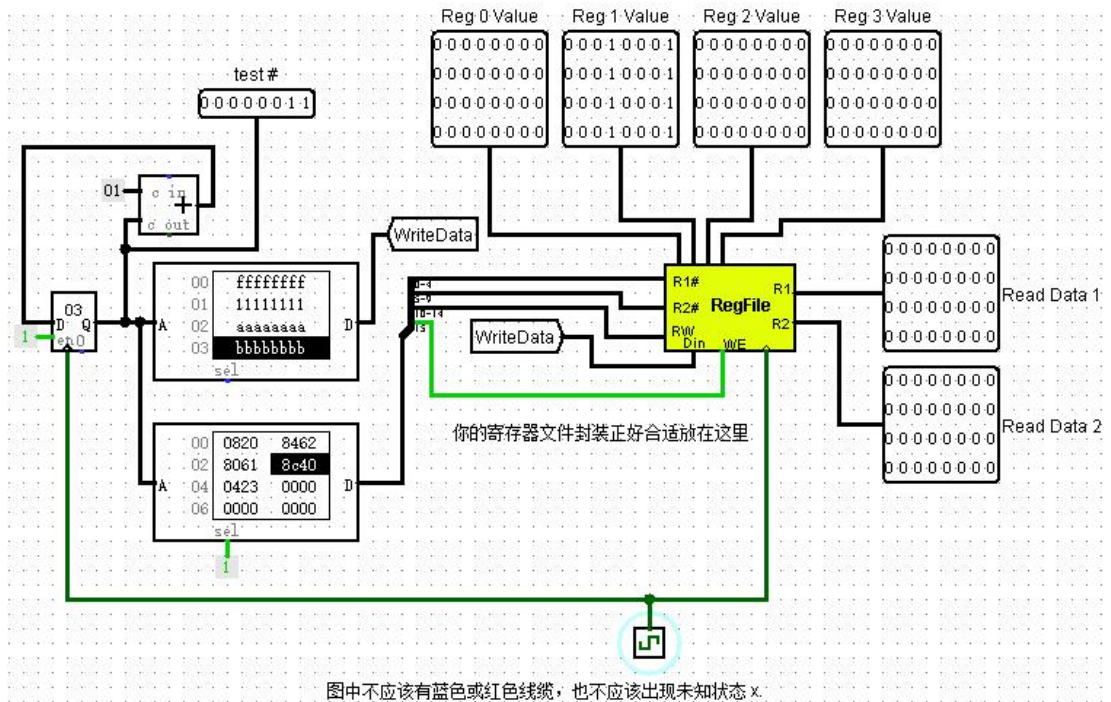
Din = AAAAAAAAAA

WE = 1

对应数据存储器值: AAAAAAAAAA

对应指令存储器值: 8061

执行完该条指令之后，结果如右图所示



MIPS寄存器文件电路测试数据及正确视图

4. 读0号寄存器和2号寄存器, 写3号寄存器

$$R1\# = 0$$
$$R2\# = 2$$
$$RW = 3$$

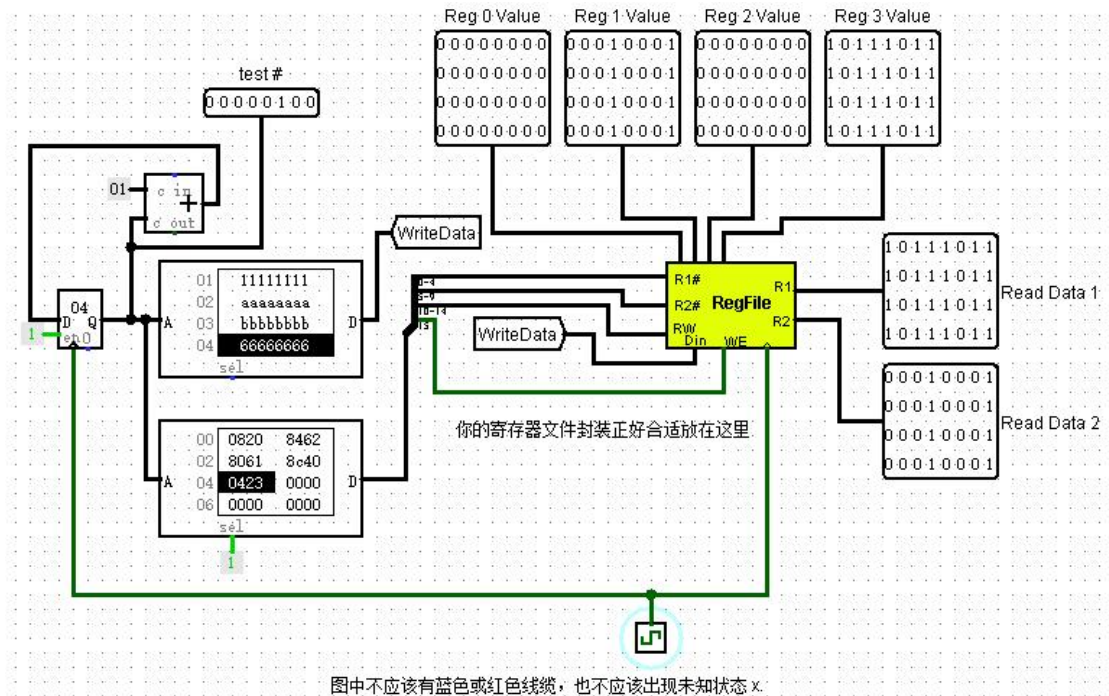
Din = BBBBBBBBBB

$$WE = 1$$

对应数据存储器值: BBBBBBBB

对应指令存储器值: 8C40

执行完该条指令之后，结果如右图所示



MIPS寄存器文件电路测试数据及正确视图

5. 读3号寄存器和1号寄存器

R1# = 3

R2# = 1

RW = 1

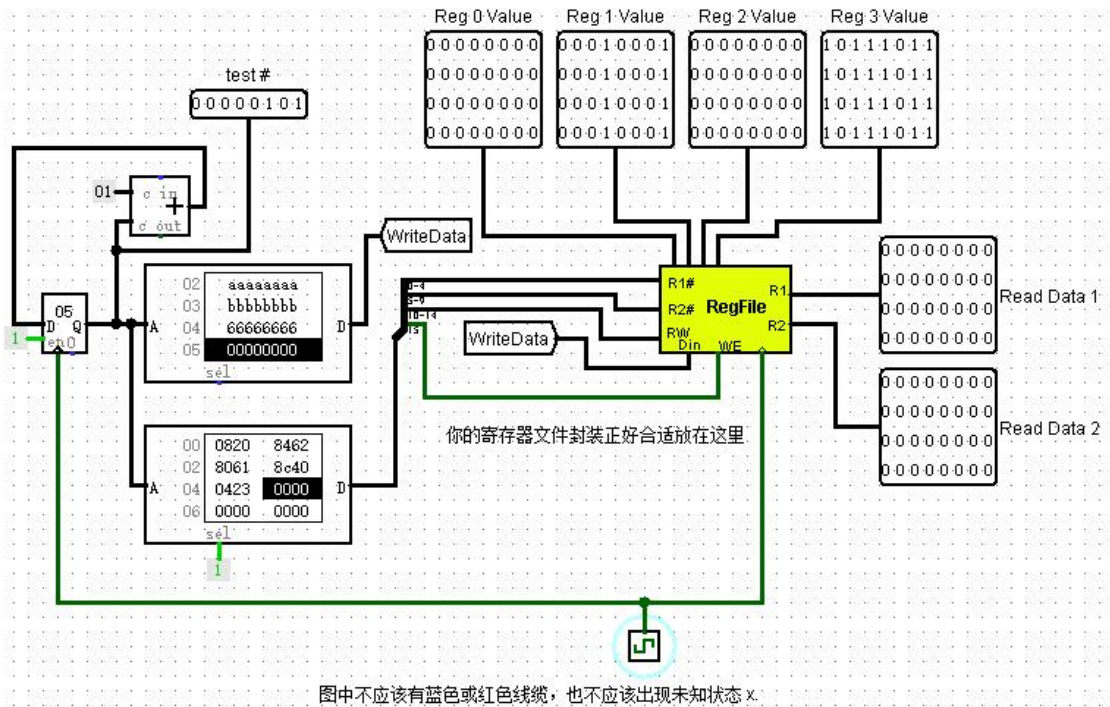
Din = 66666666

WE = 0

对应数据存储器值: 66666666

对应指令存储器值: 0423

执行完该条指令之后, 结果如右图所示





谢谢!

