单周期MIPS CPU设计

没有使用mooc提供的工具，而是直接根据educoder要求的八条指令进行设计

首先根据educoder提供的数据通路图将CPU中控制器之外的的硬布线进行连接

其次，根据硬布线控制器的要求，需要对用到的八条指令的指令译码逻辑、ALU控制信号逻辑、指令译码对应控制信号的逻辑进行设置，以下有两种方法获取具体的八条指令的func、op字段

一种方法是直接将sort.hex文件中的十六进制数据拷贝到指令寄存器中，通过指令寄存器的反汇编引脚获得八条指令与指令中func、op字段的关系

另一种方法是根据mooc中给定的MIPS指令解说，直接得到八条指令的func、op字段，并自己构造合法的MIPS指令语句，并通过反汇编引脚进行印证

通过上面两种方法，就可以完成控制器中指令译码逻辑，

然后，可以通过依次在指令寄存器中访问这八条指令，根据educoder和mooc中MIPS指令手册中解说的指令的具体含义，在CPU硬布线的基础上给定控制信号，完成指令译码对应控制信号的逻辑的设计，从而完成在不同指令下数据通路的构建

与此同时，在访问不同指令时，也可以根据指令的含义完成ALU控制信号逻辑的设计

最终，控制器得以实现，而硬布线电路已经设计完毕，从而完成了单周期MIPS CPU的设计

在搭建完成后，自己遇到了一些问题（详见电路上的备注），并通过逐条执行MIPS机器指令的方法，理解sort.hex的机器语言逻辑，并根据每条机器指令下数据通路的正确与否修改控制器设置时出现的错误

多周期微程序MIPS CPU设计

首先一个任务是完成MIPS微程序地址转移逻辑的设计，即通过机器指令的译码信号来自动生成执行该指令的微程序的入口地址，该入口地址虽然可以自己设定，但自己选择参考educoer的测试算例，毕竟写出来还是要通过这些算例的评测的，也可以防止设定地址出错而影响后面的实验

仿照mooc中MIPS微程序CPU的控制器之外的硬布线进行连接

将sort.hex数据镜像加载到存储器中，通过微指令自动生成表格，在伪指令存储器的0号单元处存放已经设定好的取指令微指令，观察一个时钟周期后，指令被成功取到IR中，借用单周期MIPS CPU的指令译码逻辑，将指令成功译码，转移地址生成逻辑生成了正确的addi指令地址，因此将指令译码微指令的P置为1，并输入表格自动生成的微指令到微指令存储器中，观察到一个时钟周期后，微指令存储器的访问地址指向正确，由于上一条微指令的其他使能均为0，因此硬布线环节中各寄存器不变

因此，可以以一个时钟周期，也就是一条微指令运行的时间为单位，参照mooc的讲义上设定的微程序控制器的状态图，调整不同指令下，为了构建正确数据通路需要的使能信号，包括ALU控制信号，并将表格中生成的16进制微指令输入到微指令寄存器中，从而逐步完成微指令寄存器的构建

如果程序运行出了问题，可以通过架构中可以设置断点的时钟信号产生器，在educoder上出现问题的地方设置断点，并在断点时刻分析数据通路中数据错误发生在哪里，并修改相应的微指令的使能信号，形成正确的时钟通路

比较特殊的是syscall指令的微程序，它只有一条微指令，并且循环执行，下址字段就是微指令自己的位置，P为0，此时其他使能信号已经无关紧要，因为微程序控制器只会循环执行这一条微指令，甚至连取机器指令微指令都不会再执行，因此微调AluOp让它的使能信号不悬空即可（原始的设定中，Alu\_Control只有三种可能的值，第四种值会令AluOp悬空，需要加以修正）

此外，由于在实验架构的规定中，syscall虽然func为0，但不属于R\_TYPE；同时，如果单纯凭借func是否为0来判断是否为R\_TYPE，就会在syscall机器指令下，同时点亮R\_TYPE和func的指令译码信号，造成微程序入口地址可能发生错误，因此对R\_TYPE的判断逻辑中要排除syscall的情况

同时，还发现，在educoder上面，有的引脚范例输出和实际输出都是大量的0，和本地的测试结果相违背，怀疑是这些位置并没有真正的获取信号，而是用了简单的下拉

多周期硬布线MIPS CPU设计

本次任务中，由于仍旧为多周期和MIPS指令集架构的CPU，控制器之外CPU布线的部分与多周期微程序MIPS CPU一致，因此替换控制器复制电路即可

同时，由于仍旧和多周期微程序MIPS CPU中，采用一样的MIPS指令集架构及相同的译码信号标签，对于指令译码和ALU控制信号的逻辑，也可复用多周期微程序MIPS CPU中，已经完成的实现。

经过mooc的提示，通过多周期硬布线CPU状态机的现态经组合逻辑生成的控制信号，也可以转化为微程序CPU中的微指令产生的控制信号，从而简化工作量

以下部分为本次实验新的内容

状态机的状态转移逻辑可以通过实验包提供的表格，自动生成表达式，从而放到logisim中自动生成电路来完成模块的完善

然后其实实验就做完了

-----------------------CPU实验更新了-------------------

在更新的CPU实验中，微程序和现代时序硬布线CPU的实验从专用数据通路改成了单总线，去掉了之前的单周期硬布线CPU实验，添加了三级时序CPU实验和附带中断的微程序和现代时序硬布线CPU

对于专用数据通路改成单总线的变化，影响主要在数据通路及微指令（控制信号）的产生上，由于老师上课使用的课件中介绍非常详细，因此按照从前CPU实验的大体思路，将微指令（控制信号）的具体值参照老师的课件进行修改，借助更新的实验框架和excel表格就可以完成

同时，这两个实验的变化还在于判别逻辑作为一个独立的模块被提出来，从前的实验中也有判别逻辑决定微程序的下址的相关操作，但是引入了外部反馈信号的判别逻辑只是作为一个小的逻辑组成，存在感不够高，这个变化在引入了中断的CPU实验中大有用处

同时，微指令（控制信号）设计表中，再次强调了beq指令的特殊，它在实际判断的微指令处需要产生一个判别位P2，它会和BEQ反馈信号一起构成判别逻辑的输入，决定下址是继续执行还是返回到取指令处，需要正确理解。

针对添加的两个实验，首先探讨三级时序的CPU实验，状态机和时序产生模块的构建十分容易，主要的用功之处在于根据指令、机器周期和时钟周期设计控制信号的产生逻辑。由于老师的PPT上对于更新的这一块内容进行了比较详细的说明，因此参照PPT即可大体完成，还可以看到三级时序漂亮的信号

一个小小的坑是SLT指令，老师没有细讲，它的用处是比较并置位，使用搜索引擎就可以找到PPT上没有的详细解说，对于数据通路的构建和add指令非常类似，可以进行微调，但控制信号的生成需要参考测试集来准确的理解题意

其次探讨加入中断的微程序与硬布线现代周期CPU实验，这两个小实验和没有周期的时候一样，微指令（控制信号）可以通用，做一个相当于做了两个。

对于中断的加入，可以借助老师的PPT以及测试集的信号进行探讨，硬布线的状态机和微指令的数量发生了变化，引入了中断处理周期和中断服务程序需要的中断返回状态，前者需要两个状态，即int1和int2，一个负责关中断并通过EPC保存断点，一个负责解析中断并将中断处理程序的地址传入PC作为新的指令开头；后者，也就是eret状态，会在中断服务程序的最后被调用，它会打开中断，并将原先EPC存放的PC旧址重新传入PC，完成断点恢复

实验设计的一个有意思的地方在于，中断服务程序和我们实际运行的冒泡程序是写在一个.hex文件中的，自己一开始没有找到，以为要自己手写到内存中，闹了笑话。因此，中断服务程序的地址也由此处提供的中断服务程序地址决定，在单总线框架中要注意地址需要左移两位来配合本框架的内存和PC++

中断的原理理解完后，可以完成表格中微指令（控制信号）的扩充，这时要注意，微程序中每个微程序最后增加了一个判别位P3，用来标识微程序（控制信号）的结束，它会和中断信号一起构成判断逻辑的输入，决定下址是回到取指令处还是进入中断周期，比较特别的是beq指令，它的判断equal的微指令（控制信号）也可以视作一个特殊的“结束”，需要将判别位置1；对于硬布线的状态机扩充，原理类似。

最后是中断的硬件搭建，这一次硬件搭建主要靠连线，其中EPC的构建最为简单，使用EPC的使能信号和三态门控制EPC寄存器数据的出入即可，注意出入最终和单总线相连

中断使能寄存器实际上表征了开中断还是关中断，开关中断的信号直接构成了它的异步置位输入，因此这个寄存器不需要时钟。它的输出可以构成中断处理模块的使能，防止中断服务程序运行的过程中被其他中断打断，这一点需要动动脑筋

中断处理模块主要负责接收、记录并输出相关的中断信号，内部逻辑可以进行简要的分析，但实验的完成只需要理解引脚，其中需要填充的是使能输入、表征中断请求存在的输出的使用、以及中断的序号输出的使用。使能输入来自上述的使能寄存器。表征中断请求存在的输出需要和中断使能信号相与后，送入控制器的中断反馈信号端口，从而保证开关中断的正常实现，防止中断服务程序被打断。中断的序号输入主要送入多路选择器，选择中断服务程序入口后通过ADDRout信号控制的三态门送入总线，供int2状态使用

至此，中断的构建基本解说完毕，由于微程序与硬布线现代周期可以一套拿下，因此带中断的CPU可以顺利完成

在实验过程中发现的一个容易忽视的问题是，填表时常常分不清输入的无关项和0项，需要对具体的组合逻辑进行分析，不能盲目的面向测试集编程，否则将本该为0项的内容空置，成了无关项，则可能导致输出冲突，比如11x产生输出11，110产生输出10，它们的输出之间就会彼此冲突，导致110下实际产生错误的输出为11。相对的，盲目的加入0项会导致组合逻辑变得更加复杂，因此填表时一定要多想。