第五讲 单周期处理器

501-处理器的设计步骤

> 处理器 或者说是CPU，是现代计算机当中最为复杂的一个部件，
>
> 不过，这并不意味着我们就做不了这件事情， 如果要设计一个简单，但是能工作的处理器，也没有那么地神秘。
>
> 在这一节，我们就一起来探索处理器是如何设计出来的。

> 要设计一个处理器，可以分为如下几个步骤：首先，是要分析指令系统，
>
> 指令系统，是在处理器设计之前，就由软件和硬件的设计人员共同协商决定的，
>
> 通过分析指令系统，我们可以得出指令所要操作的数据，需要通过怎样的一个电路的结构，
>
> 这就是数据通路；在我们得到这样的需求之后，我们就可以为数据通路选择合适的集成电- 路组件，
>
> 比如说加法器、减法器、寄存器等等； 选好了合适的组件之后，我们就按照最开始分析出的需求，
>
> 把这些组件连接起来，就构成了完整的数据通路。
>
> 但是仅有数据通路是不够的，我们还需要控制这个数据通路应该如何工作；
>
> 因此，第四步，是要分析每条指令的实现，以确定控制数据通路工作的控制信号；
>
> 最后，是把这些控制信号汇总起来，形成完整的控制逻辑，也可以称之为控制器。
>
> 那我们还是以MIPS的指令系统为例进行讲解。
>
> 当然，整个MIPS指令系统的指令还是太多了，
>
> 所以我们进行一个大幅度的简化，在我们要讲解的这个简化版本当中，包括这样几条指令。
>
> 我们提供了无符号的加法和减法指令各一条， 它们都是R型指令，然后还提供了一条逻辑运算指令，
>
> 这个指令其中一个操作数是立即数，所以它的指令格式是I型的。
>
> 这三条都是运算类指令，操作数要么是寄存器，要么是立即数，
>
> 因此我们还需要有访问存储器的指令，这就是Load指令和Store指令， 它可以在寄存器和存储器之间传送操作数。
>
> 最后是一条分支指令，我们提供了一条条件分支指令，
>
> 那我们首先就要分析这样一个指令系统，对于数据通路的设计有怎么样的需求。
>
> 首先我们通过对指令的各个位域进行分解，来看这些指令的含义。
>
> 对于R型指令，它一共分为6个位域，最高的6个比特，称为操作码，
>
> 接下去连续3个位域都是5个比特，各自标明了一个寄存器的编号，
>
> 然后5个比特在完整的MIPS指令系统当中，是用作标记移位的数量，
>
> 在我们简化后的版本当中没有使用，因此我们可以看作是保留的位域。
>
> 最后6个比特是功能位域，

因此，当我们取得一条R型指令之后，就可以将它分解为这样6组控制信号。

与之类似的是I型指令，只不过I型 指令的32个二进制位只会被切分为4组信号，

而且我们要注意，这些指令的编码都是从存储器当中取得的，

因此，我们首先需要一个存放指令的存储器，对于指令存储器 来说，它不需要支持写入的功能，只要可读就可以了，

而且我们希望对这个存储器，外界给它32位的地址，

它就会给出对应的32位的数据，那么这个32位地址又从哪里来呢?

所以这就是我们另一个需求，我们需要有一个存放指令地址的32位寄存器，

称为PC，也就是程序计数器，有了这两个组件，我们就可以取得想要的指令了。

然后我们再从指令的操作来分析其他的需求， 首先来看加法和减法指令，

这两条指令的主体功能，都是选择两个不同的寄存器，对它进行加法

和减法的运算，然后让结果存到另一个寄存器当中去。

因此，首先我们需要有一组存放数据的通用寄存器，

每个寄存器都是32位的，而且我们可以约定，这一组寄存器总共有32个，

那这样一组通用的寄存器，我们就称为寄存器堆。

从加法和减法指令的操作，我们还可以看出，

在运算时，我们需要同时读取两个寄存器的内容，这两个寄存器分别由指令位域当中的rs

和rt这两个域所指定。在运算完成后，我们还需要写入另一个寄存器，

这个寄存器的编号由rd或者rt来进行指定， 那么对于这个加法和减法运算，它都是由rd位域指定的。

而对于立即数的运算，运算结果需要改写的寄存器是由rt这个位域指定的。

而立即数运算指令的操作数，除了rs所指定 的寄存器外，另一个操作数是一个立即数，

其中16位是直接填写在指令位域当中的， 但是我们的运算需要是32位的，

因此，我们还需要一个功能，就是将16位的立即数扩展到32位，对于这个运算，

我们需要的扩展方法是零扩展，也就是将 高16位都填0，从而构成一个32位的数。

那这三条都是运算指令，因此我们还需要支持不同的运算类型。

在这里我们可以看出，我们需要提供加法、减法和逻辑或三种功能。

因此，我们还需要一个这样的运存器，这个运算器的操作数可以是两个寄存器，

也可以是一个寄存器加一个扩展后的立即数，这些就是运算指令的需求了。

然后我们再来看访存指令，对于LOAD指令 来说，它需要从存储器当中读出一个字，

而这个字所在的存储单元的地址，是由一个寄存器的内容 加上一个立即数来决定的，取出这个字之后，

会把它存放到寄存器堆当中，由rt所指定的寄存器。

与之相对的还有STORE操作，STORE操作，则是将rt 所指定的寄存器的内容，传送到内存的指令的存储单元中。

那对于这两条访存指令，我们又有什么需求呢？首先我们需要一个能够存放数据的存储器，

这个存储器既要可读，也要可写，它的地址 输入，以及输入和输出的数据，都应该是32位的。

另外我们从这个地址的计算方法还可以看出，我们也需要将16位的立即数扩展到32位，

但扩展方法是符号扩展，也就是将这个立即数作为低16位，

并将其最高位复制到高16位当中去，从而形成一个32位的立即数，

那这就是访存指令的主要需求。最后我们来看分支指令，

对于分支指令，首先要判断两个寄存器当中的内容是否相等， 如果相等，那就将指令位域中立即数的这一部分

经过一个简单的变化，加到PC上，从而得到新的PC，

这个变化的规则和原因，在介绍MIPS指令系统时，都已经讲解了，所以在这里，

我们可以简化地认为，就是将当前的PC加上了一个立即数，从而得到了一个新的PC。

但如果刚才那次比较的结果是不相等， 那就直接将当前的PC加上4，从而产生新的PC。

因此，分支指令的需求，首先是要能够比较两个寄存器的内容，并判断

是否相等，然后还需要PC寄存器支持两种自增的方式，

一种是加4，一种是加一个指定的立即数，

当然，对于PC加4这个需求，前面介绍的其他指令也都是需要的。

那我们再把刚才分析的这些需求整理一下。

首先我们需要一个算数逻辑单元，也就是ALU，它要支持

加法、减法、逻辑或和比较相等这样的操作；

它有两个32位的输入，可以来自寄存器，也可以来自扩展后的立即数；

然后我们还需要一个立即数的扩展部件，可以将一个16位的立即数扩展为32位，

而且扩展方式可以是零扩展，也可以是符号扩展；

我们还需要一个程序计数器，这是一个32位的寄存器， 由时钟控制，而且我们还要给它支持两种加法运算，

要么加4，要么加一个立即数，这样的需求我们可以用ALU来实现，也可以只给它配上

简单的加法器。除此之外，我们还需要两个带有存储功能的组件，

一个是寄存器堆，一个是存储器，对于寄存器堆， 我们一共需要32个32位的寄存器，

需要支持同时读出两个寄存器，和写入一个寄存器， 这样的寄存器堆特征，我们称为“两读一写”。

而对于存储器，我们需要一个只读的指令存储器，地址和数据都是32位的，

还需要一个可读写的数据存储器，地址和数据也都是32位的。

我们从这个需求本身来看，实际上提供一个可读写的存储器，就可以满足指令和数据的需求，

而且这也是符合冯诺依曼结构对于统一的一个存储器的要求。

那么在现代计算机当中，内存就是这样一个指令和数据统一存放的存储器，

但是在处理器内部，现代的设计往往会设置高速缓存， 也就是Cache这样的部件，用来保存内存当中的一部分数据。

那么高速缓存这个结构，是会被分成指令和数据 两个部分，因此在这里，我们也选择了分开的结构，

但我们要注意的是，这个存储器我们实际上对应的是CPU 当中的高速缓存，而不是整个计算机当中的内存。

我们再来分别看一看这两个组件的具体需求。

这就是一个寄存器堆的结构示意图， 内部有32个32位的寄存器，它有三组数据接口，

其中busA和busB是两组32位的数据输出接口， busW是一组32位的数据输入接口，

这也体现了“两读一写”的特性。那么如何对寄存器堆进行读写呢?

那就需要来看几组读写控制信号。首先是Ra，这是一个5位的信号，

5位的信号，正好可以选择编号0-31的寄存器， 因此，寄存器堆会根据Ra的输入，选择

对应编号的寄存器，将其内容放到busA信号上； 然后是Rb信号，同样，寄存器堆也会根据Rb的输入，

选中其对应编号的寄存器，将内容放到busB信号上。

这样外界将两个编号分别放到Ra和Rb的信号输入中，寄存器堆

就会将对应寄存器的内容，分别放到busA和busB的信号线上， 这就完成了同时读取两个寄存器的功能。

而对于写，稍微复杂一些，首先，将要写的寄存器的编号

通过Rw信号先输入，在时钟的上升沿，如果写使能信号是有效的， 也就是WriteEnable信号等于1，

那么寄存器堆就会将busW信号上的内容存入Rw信号所指定的寄存器，

这就是寄存器堆所提供的写的功能。我们需要注意的是，寄存器堆的“写"是在时钟上升沿- 完成的，

但是寄存器堆的"读"是不受时钟控制的，也就是说，在任何时候，就让Ra 或者Rb发生

了变化，那对应的busA和busB的信号就会发生变化。

然后我们来看存储器，存储器有两组 数据接口信号，有32位的数据输入和32位的数据输出。

那存储器的读写实际上和寄存器堆有些类似，只要我们给入一个地址，

存储器就会将对应的存储单元中的数送到数据输出信号上。

而与寄存器堆不同在于，我们只给入一组地址信号， 而不是寄存器堆所提供的两组寄存器编号的输入。

从这个意义上，我们也可以说，这个存储器是一个易读易写的存储器。

另一个方面，我们给入的地址信号是32位的， 所以理论上，这个存储器可以达到2的32次方那么大，

当然这只是一个理想化的情况。

那么对于存储器的写，我们也提供了一个写使能的信号，

那在时钟上升沿到来的时候，如果写使能信号为1，那么存储器

就会将数据输入信号的内容存入地址信号所指定的存储单元。

同样我们也需要注意，存储器的”写“操作，是在时钟上升沿的时候发生的，

而存储器的"读"操作，则不受时钟信号的控制。只要输入的地址信号发生

变化，经过很短的访问时间，数据输出信号就会随之发生变化。

那现在，我们就完成了指令系统的分析，

得出了对数据通路的需求，并且为数据通路选择了合适的组件， 之后就可以开始着手建立数据通路的工作了。

在这一节，我们一起学习了设计一个处理器所需要的主要步骤，

并且构造了一个简化后的指令系统，并对其进行了细节的分析，

从而完成了设计处理器的准备工作。

502-数据通路的建立

现在一方面我们已经有了指令系统的总体需求，

另一方面，我们也准备好了几个关键的组件，

那现在我们就来一起看一看能否根据这些需求，将这些关键的组件拼合起来， 构造出一个完整的数据通路。

要建立一个

数据通路，基本原则是分析指令系统当中每一条指令的需求，并根据

指令的需求连接我们已经选好的组件。那么指令的需求

又分为两大类，有一些需求是所有指令所共同需要的； 另外，不同的指令也有一些各自特殊的需求。

首先，我们来看所有指令的共同需求，我们要执行指令，

首先就需要取指令，我们知道，指令是放在指令存储器当中的，

而我们要访问指令存储器取得指令，就得有一个地址，这个地址则存放在PC寄存器当中。

我们已经有了一个32位的PC寄存器，那我们把PC寄存器 的输出就作为指令的地址，连接到指令存储器，

而指令存储器则会根据地址的输入选中对应的存储单元，并将其内容输出，

这样我们就得到了我们所需的那条指令的二进制编码。

那除了取得当前的指令之外，我们还得为取下一条 指令做好准备，这就需要更新PC寄存器，

这又分成了两种情况，大多数时候，指令是顺序执行的，

在这种情况下，PC只需要加上当前这条指令的长度，就可以得到下一条指令的地址了。

在MIPS指令系统当中，每一条指令都是4个字节的，所以只需要

简单地进行PC加4的运算，那在我们刚才的这个结构上面，

我们需要增加一个简单的加法器，其中一个输入是PC寄存器的输出，

另一个输入是一个常数4，然后我们将这个加法器的输出连接到PC寄存器的输入，

这样，当前PC寄存器的内容既会送到指令存储器，以获得指令的编码，

也会送到加法器的输入，从而计算出一个PC+4的值， 那在下一个时钟上升沿到来的时候，

PC寄存器就会将PC+4的值存入其中，

然后再将这个更新后的内容，同时送到指令存储器和加法器，

如此周而复始，就完成了每个时钟上升沿时，更新PC寄存器

的内容，然后指令存储器就会送出新一条指令的二进制编码。

当然我们这说的是顺序执行时的情况，而如果遇到分支指令，

那下一条指令的地址就不是简单的PC+4，而是用分支指令进行指定。

因此我们还需要继续修改这个结构，我们需要增加一个二选一的多选器，

在顺序执行时，我们选择这个多选器的0号输入端，也就是PC+4的内容，

而在发生分支时，我们就选择这个多选器的1号输入端，以至于由分支指定的目标地址。

那在下一个时钟上升沿到来的时候，PC寄存器就会采样这个多选器的输出，

并将其保存起来，那这样一个结构，就完成了不断取回指令的功能。

我们把它称为取址部件，也简称为IFU，

IFU作为一个整体，从外界只有一个时钟信号的输入， 和一个多选器选择信号的输入，并且提供一个指令编码的输出，

我们只用在系统启动时，给PC寄存器一个合适的初始值， 并在指令存储器中存放好我们需要运行的指令，

然后在运行的过程中，给出合适的多选器的选择信号，这个 IFU就可以在时钟信号的驱动下自动的连续工作

起来了，这些就是所有指令的共同需求。

然后我们再根据指令的不同类别，来分析他们的各自需求。

首先我们来看加法和减法指令， 这两条指令都是R型指令，它的主体操作是先用rs

和rt，这两个5位信号，访问寄存器堆获得两个寄存器的内容，

并对他们执行相应的运算，这个运算目前我们提供了加法和减法两种，

然后将运算的结果放到另一个5位信号rd所指定的寄存器当中。

因此，我们就需要用到寄存器堆这个组件， 这是一个两读一写的寄存器堆，

两个需要读出的寄存器的编号，是由指令编码当中rs和rt这两位域指定。

而要写入的寄存器的编号，则由rd这个位域指定的。

所以我们只需要把这三个位域的信号的值，连接到 对应的输入上，这样在寄存器堆的输出端，

busA就会输出rs所指定的寄存器的内容， busB则会输出rt所指定的寄存器的内容。

然后我们将busA和busB连到ALU的输入端， 并且我们根据指令编码当中的操作码

和功能位域，就可以知道当前是加法还是减法指令， 我们要通过一个控制信号，来选择当前ALU提供的运算的类型，

然后我们还要将ALU的输出连接到寄存器堆的输入端，也就是busW信号，

那在下一个时钟上升沿到来的时候，如果寄存器堆的写使能信号是有效的， 在这里，就用这个RegWr信号等于1的时候，

寄存器堆就会采样busW信号上的内容，并 将其存入到Rb这个信号所指定的寄存器当中去。

这样就在一个时钟周期完成了一条加法或者减法的指令，我们要注意的是，这里用

红色表明的这两个信号，都成为控制信号。然后我们来看逻辑运算指令的需求，

在我们的简化版本当中，提供了逻辑运算，就是这个对立即数进行或操作的指令，

这条指令是一条I型指令，那我们刚才已经 建立的这个数据通路，是否能满足逻辑运算指令的需求呢?

经过分析，我们可以发现几个问题：第一，逻辑运算指令，它的目的寄存器

是rt，而不是rd，而我们现在在寄存器堆的Rw输入端连接的是rd这个位域，

这样就没法满足这条逻辑运算指令，要把运算结果 写入到rt所指向的寄存器当中，这样一个目的;

第二，这条指令其中一个操作数是一个立即数， 而我们现在ALU的两个输入都是来自寄存器堆，

对这一点也无法支持；第三，我们在指令当中，只提供了一个16位的立即数，

而ALU的输入都是32位的，所以还需要对这个16位的立即数进行零扩展。

针对这个三个问题，我们就需要对现在的数据通路进行改造， 我们来注意这个数据通路的图，看我们需要增加哪些部件。

针对问题1，我们增加了一个二选一的多选器，当执行之前的加法或者减法指令时，

我们让多选器的选择信号为1，这时就会和刚才一样，将 rd信号的内容传递到寄存器堆的Rw信号端，

而在执行现在这条逻辑运算指令时，我们就可以选择这个多选器的零号输入端，

将rt这个信号的值传递到寄存器堆的Rw信号端， 这就为写入rt所指定的寄存器提供了支持。

针对第二个问题，我们在ALU的输入端增加了一个多选器， 对于之前的功能，我们通过这个多选器的零号输入端，

将寄存器堆的busB信号和ALU的输入端相连，

而对于这条逻辑运算指令提出了新的需求，我们可以通过这个多选器的一号输入端，

将立即数与ALU相连，当然，这个16位的立即数还需要通过

一个零扩展部件，扩展成32位，再接到这个多选器的输入上。

这样我们就通过增加了两个多选器和一个零扩展部件，来满足了逻辑运算指令提出的新需求。

我们需要注意的是，我们在满足新需求的时候，一定要保证原有的需求同样是得到满足的。

所以在这里，我们就需要增加多选器，而不是直接修改各个部件的输入。

那在现在这条数据通路的基础上，我们再来看访存指令的需求。

访存指令也是I型指令，我们先来看其中的Load指令， Load指令是要对数据存储器进行访问，

而访存的地址是由rs所指定的寄存器的内容加上立即数进行符号扩展后的值，

而且访存得到的数据也要保存在rt所指向的寄存器当中。

那现在的数据通路，已经支持了写入rt所指向的寄存器了。

而访存地址的运算， 是一个寄存器加上一个立即数， 那我们现在的数据通路，也大体支持这样的一个功能，

但仍然还是有问题的。首先，我们还不支持符号扩展，

其次，这个ALU运算的结果，应该是作为地址去访问

> 存储器，从而获得数据，而不是直接连到寄存器堆的写入端， 因此我们还需要对这个数据通路进行改造。
>
> 针对符号扩展的需求，我们将原先的零扩展 的这个功能部件，改造为一个多功能的扩展部件，
>
> 通过控制信号，我们可以选择进行零扩展或者进行符号扩展，
>
> 对于这条load指令，我们可以选择将16位的立即数符号扩展为 32位的数，并通过这个多选器连接到ALU的一个输入端，
>
> 而ALU的另一个输入端，则是rs所指定的寄存器的内容，
>
> 然后执行加法运算之后，获得了存储器的地址， 我们在这里就需要新增一个数据存储器，
>
> 这个存储器根据地址就可以得到对应的存储单元中的数据，
>
> 因为我们最终是要将这个数据写入到寄存器堆，所以还需要增加这样一个多选器，
>
> 将数据存储器当中输出的内容传送到寄存器堆的输入数据端，
>
> 那这就是load指令所提出的需求，我们的解决方案 是将原有的零扩展部件增加符号扩展的功能，并增加
>
> 一个数据存储器，当然，还包括相应的多选器。
>
> 那我们还有另一条访存指令，那就是Store指令，
>
> Store指令的地址运算方式和Load指令是一样的，
>
> 所不同的是，Store指令是将rt所指定的寄存器当中的内容存放到数据存储器当中去。
>
> 那我们来看现在的这个数据通路，我们首先需要rt所指定的寄存器的内容，
>
> 在现在的这个数据通路中，rt所指定的寄存器的内容会从busB信号上出来，
>
> 那我们就需要将这个信号连接到数据存储器的数据输入端，
>
> 但是对于除了Store之外的其他指令，我们都不希望将busB上的信号写入到数据存储- 器当中，
>
> 所以我们还需要给数据存储器连接一个控制信号， 只有在这控制信号有效的时候，才会进行写入的操作，
>
> 这样我们就满足了Store指令的需求。
>
> 现在，除了比较特殊的分支指令之外，我们分析了其他指令的需求，
>
> 并将各个组件连接起来，再加上之前已经构造的IFU部件， 我们就初步完成了数据通路的建立工作。

> 现在，在处理器的设计步骤的这五步当中，我们已经完成了第三步。

> 现在，我们已经基本完成了一个数据通路，看起来
>
> 也并不复杂嘛。不过，这个数据通路是否已经完整了呢？ 而它又是否能够正确地工作呢？
>
> 这些都要通过一条一条指令来进行更细致的检查
>
> 和确认。

503-运算指令的控制信号

> 通常，在一个指令系统中，运算指令是最为重要的。
>
> 而从另一个方面看，一般来说，运算指令的执行过程是最为简洁明了的。
>
> 所以我们先来看，运算指令所需的控制信号，是怎样构成的。

MIX 的指令系统，从功能上划分

可以分为运算指令，访存指令，和分支指令。

而从指令在格式上划分，只可分为R型指令，I型指令，J型指令。

那在我们现在用的这个示例中，经过简化之后 只有六条指令，分布在了这个表格的这四个地方。

那首先，我们来看运算指令当中 R 型格式的指令，就是加法指令和减法指令。

这两条指令的操作过程非常的相似，因此我们就用其中的加法指令作为例子进行讲解。

加法指令的操作可以分为三个 步骤，首先是从指令存储器当中取回指令，

然后将 rs 指定的寄存器和 rt 指定的寄存器 的内容相加，运算结果保存在 rd 所指定的寄存器当中。

最后是计算下一条指令的地址。

执行完加法指令，下一条指令是顺序执行的，因此将 pc+4 更新到pc 当中。

那么就依次来看这三个步骤， 首先是从指令存储器中取回指令，

那么这个步骤并不是加法指令特有的，对于所有的指令都需要有这个 步骤，这步操作是在取指部件，也就是

IFU 当中完成的。

我们假设在系统中的某一个时刻， 我们就选取 T0 所示的这个时钟上升沿之后很短的一段时间。

那这时 PC 寄存器的输出已经稳定，它的输出就是当前要取回的这条指令的地址。

这个信号会连接到指令存储器的地址输入端。

那经过一小段访问时间之后， 指令存储器就会将这个地址所指定的存储单元的内容放在其输出端。

也就是我们要取回的指令的二进制编码。

我们用红色标出了这个步骤所关注的有效的信号。

同时我们注意左边的这个时钟信号。

我们用以标识当前时刻的这条虚线也向后移动了一小段。

现在我们已经有了这条指令的二进制编码，我们就可以进入第二步。

第二步是执行加法运算，但我们还得先取得操作数，

操作数所在的寄存器的编号，是放在指定编码中的。

现在 IFU 已经送出了指令编码。

我们根据 R 型指令的位域分布可以从 IFU 输出的这 32 位信号中选取对应的部分。

例如，第二十五位到二十一位组成一组信号，我们记为 rs，第二十位到十六位我们记为 rt。

第 15 位到 11 位我们记为 rd。

那这三个信号就会被连接到寄存器堆的输入端。

我们注意看左边的寄存器堆，rs 信号被连接到寄存器堆的 i 输入端。

那么注意实际上这根标了 rs 的信号和最右边这根标了 rs 的信号是连在一起的。

只是为了避免这张图显得过于杂乱，我们没有把这条线画出来。

以此类似，rt 和 rd 这两个信号， 也都分别连到了这张图左边标了 rt 和 rd 的地方。

那根据寄存器堆的设计特性， 它随时会根据 ra 和 rb 这两个端口输入的信号，

找到对应编号的寄存器，并放在输出端。因此一旦 IFU 取回了这条加法指令

那其中 rs 所对应的寄存器的内容就会放到上 busA 信号上， rt 所对应的寄存器的内容就会放到 busB 信号上。

那接下来对这两个数应该做怎样的操作那就由这些标成红色的控制信号来决定了。

首先我们来看给 IFU 的这个控制信号，

因为这是一条加法指令，所以我们给出下一个 PC 的选择信号。应该是选取 pc+4 的这个通道。

我们暂且用 +4 来表示 然后我们来看寄存器堆的这两个输出，busA 已经被直接连到了 ALU 的一个输入端。

而 BusB 是连到了一个多选器，这个多选器负责为 ALU 的另一个输入端选择输入信号。

那对于这条加法指令，我们正是需要从寄存器堆输出的这个 BusB 信号。

所以我们来看这个多选器对应的选择信号它就应该是零。

那既然如此，这个多选器一号通道的输入的信号，就没有意义了。

而产生这个信号的是立即输的扩展部件。

这个部件也有一根控制信号，决定了它是进行零扩展还是符号扩展。

那现在这个控制信号设成什么都可以，我们用 x 来标记。

那好，现在 ALU 的两个输入端都已经连接上了正确的信号。

那么还需要给 ALU 一个控制信号，以便让 ALU 执行加法操作。

这个控制信号，我们就以 ADD 来标记。那经过一段运算的时间之后， ALU 就会输出这两个数相加的结果。

这个运算结果的信号会送到两个地方，其中一个是数据寄存器的地址输入端。

但是我们通过这条指令的分析我们可以知道，这条加法指令是不需要读写数据存储器的。

因此，我们首先需要把数据存储器的写使能信号作为无效。

否则在下一个时钟上升沿， 数据存储器会采样它的数据输入，从而改变其中的内容。

这就不是我们希望达到的结果，但即使把写使能信号置为零

数据存储器依然会根据地址的输入选择对应的存储单元并将其内容送到数据输出的信号上。

因此我们还需要正确的设置最后这个多选器的控制信号。

现在我们需要把它设成零。

这样这个多选器就会把零号通道的内容，也就是 ALU 的运算结果放到这个多选器的输出上。

那这个信号最终会被连到了寄存器堆的输入端。

而我们希望在下一个时钟上升沿到来的时候，寄存器堆会 采样其输入端的信号，并保存到 rd 所指定的寄存器当中去。

因此，我们首先需要将寄存器堆的写使能信号置为有效， 而要写入的寄存器的编号我们需要在

rd 和 rt 当中选一个。

根据这条指令，我们应该选择 rd，所以这个多选器的控制信号应该设为一。

那这样所有的控制信号都已经设置完成了。

而时间也过去了一小段，我们从这个时钟的波形上也可以看出来。

那只要所有的控制信号都设置正确了。

从寄存器堆输出的这两个寄存器的值就会经过 ALU 完成加法。

然后经过这个多选器，最终送到寄存器堆的输入端。

那我们也将在这个步骤中涉及的有效的信号用红色标出来。

那么注意到，又过了一小段时间之后， 寄存器堆输入端的信号就会变得稳定。

那在这之后，当时钟上升沿到来的时候，寄存器堆就会正确的采样 busw 的信号。

并选中 rd 所指定的寄存器，将采样的信号保存在其中， 这样就完成了这条加法指令这一步的操作。

不过这条指令的工作还没有完成，我们还有第三步，就是更新 pc 寄存器的内容。

除了分支指令，其他指令都应该是顺序执行的。

因此都应该用当前 PC 寄存器的值加上 4 作为下一条指令的地址，我们注意这个时钟信号的波形，

虽然我们这里说是第三步，实际上它和刚才第二步完成的时间是一样的。

我们来看对应的 IFU 的结构图，其实我们在第一步当中提到的， PC

将其输出送到了指令存储器的地址输入端， 与此同时 PC 的这个输出也会送到加法器的输入端，

加法器的另一个输入端固定连了常数 4， 这个加法器的输出就会计算出 PC+4。

而当指令取出之后，经过一小段时间，就会产生对应的控制信号。

对于这条加法指令，这个 nPC_sel 的信号就会被设为零，也就是

我们刚才标记的 +4，那它所控制的这个多选器就会选中零号通道。

也就是将 PC+4 送到输出并最终连到了 PC 寄存器的输入端。

那在下一个时钟上升沿到来的时候，也就是这个图中标记的 T1

的时刻， PC 寄存器就会采样它的输入，从而把 PC+4 存到寄存器当中。

那再过一小段时间，也就到了 T1 这个上升沿之后 的一小段时间，这个 PC 寄存器的输出就会稳定。

从而把递增了4 之后的地址再次送到了指令存储器取出了下一条指令。

这样这个取指在并执行然后再取指再执行的过程

就会自动的进行下去，而且每一条指令都是在一个时钟周期之内完成的。

我们再来看另一类 指令，也就是运算指令当中的

I 型格式的指令， 在我们现在所用的这个指令系统当中，这一类指令只有一条。

我们就来看这条指令的操作步骤，它同样也分为三步，

第一步就是从指令存储器中取回指令，第二步则是进行逻辑或运算， 这个运算的一个操作数是由指令编码中

rs 域所指的一个寄存器。

另一个操作数则是指令编码中的 16 位立即数经过零扩展之后得到的 32 位数。

那这个或运算的结果最终会保存到指令编码当中 rt 位域所指定的寄存器中。

然后第三步是将当前 PC 寄存器的值加上 4，再更新到 PC 寄存器当中去。

那第一步和第三步不我们刚才介绍的 I 的指令 是一样的，因此我们只来看其中的第二步，

那这个表达式就是第二步操作的描述，我们结合这个数据通路来看。

刚才我们已经在 IFU 输出的指令编码中选出了三组信号， 那现在我们还需要再多一组信号，也就是第

16 位，我们把它组成一个 16 位的立即数这个信号。

那这个信号与我们之前连出的这三组信号一样也会被连接到数据通路上。

那我们注意，这个信号会被连接到这里。

这 16 位立即数就会被连接到扩展功能部件。

那对于这条指令，这些控制信号，又应该如何设置呢？ 首先我们还是来看

IFU，对于这条 OI 指令，它的下一条指令，依然应该是 顺序执行的，所以

NPC_sel 信号也应该被设置为 +4，这和 ADD 指令是一样的。

那这条指令与 ADD 指令最大的不同在于， ALU 的第二个操作数是一个立即数，而不是来自于寄存器堆。

所以归 ALU 的第二个输入端选择来源的多选器，它的控制信号现在就应该是 1。

选择立即数作为来源，而这 32 位 的立即数是由指令编码当中的 16 位立即数扩展而得的。

根据这条指令的要求我们应该采用零扩展的形式。

所以这个扩展功能不接的控制信号，应该被设置为零扩展。

然后我们再来看，ALU 的运算功能 选择信号。我们应该控制 ALU 进行或运算。

然后我们来看数据存储器的写使能信号。因为这条 指令不需要写数据存储器，所以这个写使能信号应该被设为零。

然后我们再来看最右边的这个多选器， 那我们现在应该是将 ALU 的输出，也就是这个或运算的结果。

送到寄存器堆去，所以这个多选器应该选择零号通道。

那这样 ALU 的运算结果就会最终连接到了寄存器堆的输入端。

与此同时我们还要设置寄存器堆的写使能信号有效，而且从这个指令的要求

可以看出目的寄存器是用 RT 这个位域指定的。

所以对于寄存器堆写入寄存器的编号，我们应该选择 rt 这个来源。

因此，需要将这个多选器的选择信号设为零号通道。

这样我们所有的控制信号都设置好了。

在这些控制信号的控制下，有效的数据通路就会呈现这样一个状态。

最后是 rs 位域所指定的一个寄存器碰到了 ALU

而 16 位的立即数经过零扩展之后也送到了 ALU， 这两个数在

ALU 中进过了或运算之后， 通过这个多选器最终送到了寄存器堆，然后在下一个时钟的上升沿，

寄存器堆会采样这个或运算的结果。

然后选择 rt 位域所指定的寄存器，把这个运算结果写进去。

当然我们也要记得，在同一个时钟的上升沿， IFU 中也会将

PC 寄存器更新为 PC+4， 这样我们就在一个周期中完成了这条 OI 指令的操作。

504-访存指令的控制信号

对于MIPS这样的指令系统来说， 其运算指令是不能直接访问存储器的，

因此，我们还需要设计单独的访存指令， 用来完成寄存器和存储器之间的数据传输。

在这一节，我们就来看一看这一类访存指令是如何生成控制信号的。

对于我们要实现的这个处理器，访存指令都是属于I型指令， 一条是load，一条是store，我们分别来进行分析。

对于load指令， 首先，当然也是取指；然后根据运算出的地址，访问

数据存储器，并将读出的内容写入到rt指定的寄存器中； 最后，是更新PC寄存器。

那第一步和第三步，与运算指令都是一样的，在这里，我们就只讲解第二步相关的操作。

这是我们的数据通路，如果当前IFU

取回的是一条load指令，那这些控制信号将会如何设置呢?

首先，下一条指令的地址应该还是采用PC+4的方式。

然后我们来看寄存器堆这一边，rs和rt都是固定连接到了指令编码的对应位域，

因此，busA和busB这两组信号上，分别是由rs和rt所指定的寄存器的内容。

但我们要注意，对于这条指令，我们想要运算的是rs寄存器中的内容和立即数的符号扩展进- 行加法。

因此，对于ALU的第二个操作数的来源，

我们应该通过这个多选器，选择通道1，并且设置扩展部件为符号扩展的功能。

这样，指令位域中的16位立即数就会经过符号扩展，最终连接到了ALU的第二个输入端，

而我们发送到ALU的控制信号，则是要设置为加法运算。

这样，ALU就会完成了访存地址的运算， 并将地址信号送到数据存储器的地址输入端。

同时我们也要注意，busB上的信号，也会被连接到数据存储器的数据输入端，

虽然我们并不需要它，但是这根信号仍然会把Rp所指定的寄存器的内容送过来。

所以我们需要设置数据存储器的写使能信号， 把它设为0，保证了数据存储器的内容不会发生改变。

然后我们来看最后这个多选器， 这个多选器的两个输入，一个是ALU的运算结果，就是访存的地址；

另一个是从数据存储器当中，读出的数据。

那么对于这条指令，我们显然希望把后者送到寄存器堆当中去， 因此对于这个多选器来说，我们需要设置它的选择信号为1，

这样我们就可以把数据存储器的输出传送到寄存器堆的数据输入端， 而且因为我们要写入rt所指定的寄存器，

所以我们需要设置寄存器堆的写使能信号为有效， 并把写入寄存器编号的来源设置为rt，

这样，在下一个时钟上升沿到来的时候，

数据存储器输出的内容，就会被写入到rt所指定的寄存器当中去了，

而且在同样的这个时钟上升沿，PC寄存器的内容也不会被更新为PC+4，

这就是load指令所完成的操作。

然后我们来看store指令，

store指令也分为三步，同样，我们也只看其中的第二步。

对于store指令，我们也需要将rs寄存器中的内容和立即数

的符号扩展进行加法运算，并以此作为地址访问数据存储器， 不过不是读出，而是写入，

因此，对应到数据通路上，我们就直接标出了对于store指令，有效的这些信号。

那这些控制信号又该如何设置呢？ 那你不妨自己想一想，然后我们一起快速地浏览一遍。

对于IFU，我们要选择下一条指令的地址，

仍然是PC+4，我们要选择ALU的数据的输入， 现在我们选的是立即数的符号扩展，

而对于ALU，我们依然要选择加法的运算类型，

ALU的运算结果，依然是一个地址，会被送到数据存储器上，

但是与之前不同的是，这条指令要完成向数据存储器的写操作，

所以在这里，我们会把数据存储器的写使能信号置为有效。

这样，在下一个时钟的上升沿，数据存储器

就会采样其数据输入端的信号，但这个输入端的信号又是什么呢？ 我们可以看一看，实际上它连到了busB，而

busB是由rt这个信号所指定的寄存器中的内容。

因此，在下一个时钟上升沿到来的时候， 数据存储器会根据ALU算出来的地址，选中对应的存储单元，

并采样这个来自rt寄存器当中的内容， 将它存到对应的存储单元中去。

对于store指令来说，它就完成了它所需要做的主体的操作，

但是对于这个数据通路，我们必须要把剩下的控制信号都设置完整。

我们来看后面这个多选器，因为对store指令，我们不需要改写寄存器堆的内容，

所以无论选择哪个通路，都是没有意义的。

在这里，我们就设置这个控制信号，可以是0，也可以是1，就用x来进行标记。

那这样，这个多选器总会选出一组信号， 最终送到寄存器堆的数据输入端。

不过，这没有关系，我们只需要设置寄存器堆的写使能信号为无效，

在时钟上升沿到来的时候，寄存器堆的内容就不会发生改变。也正因为如此，

这个要写入的寄存器编号，无论设成rb还是rt，也都是可以的，

这就是对于store指令的控制信号的设置方法。

现在，我们已经掌握了访存指令是如何生成控制信号的，

再加上之前的运算指令，我们就可以编写程序，让计算机完成相应的任务了。

> 不过，如果我们想编写出功能更为强大的程序，
>
> 那只有运算指令和访存指令是不够的，我们还需要改变程序流向的分支指令，
>
> 这一点，我们在下一节继续分析。

505-分支指令的控制信号

> 分支指令，是一类特殊的指令，
>
> 它能够改变程序的流向，因此，想要执行分支指令， 我们还需要对现有的结构，进行进一步的改造。

> 在我们现在这个事例的指令系统当中，分支指令只有一条，
>
> 它的格式是I型的，那我们首先来看一看分支指令是如何工作的。
>
> 左边是一段C语言的代码，是一段典型的if else语句，
>
> 那我们看对应如何产生MIPS的汇编语言代码，会是怎么样的。
>
> 首先就是一条beq的分支指令， 看来编译器已经把 i
>
> 和 j 这两个局部变量 分别放到了s3和s4这两个寄存器当中，
>
> 那么这条指令就是比较s3和s4的值， 如果它们相等，则会跳转到True这个标号所标明的地址。
>
> 那么这里是一条加法指令，这条指令就是执行了f=g+h这条语句，
>
> 也就是C语言代码当中，if条件为真时，所要执行的语句。
>
> 执行完这条语句之后，程序将顺序地执行后面的内容。
>
> 而如果if语句的判断条件不成立，那对应的这条beq
>
> 指令在执行时，也会发现s3和s4寄存器的内容不相等， 从而不发生分支转移，而是执行顺序的后一条指令。
>
> 后一条指令是一条减法指令，
>
> 那与刚才我们看到的那条加法指令相对应，实际上，它执行的就是f=g-h这条语句，
>
> 也就是C语言当中，else的这条语句。
>
> 那么执行完这条指令之后，下一条指令是一条无条件的转移指令，
>
> 直接跳到Next，那这就是应用条件分支指令的实例。
>
> 那我们就来看一看beq这条指令的控制信号是如何生成的。
>
> beq指令的操作，同样也可以分为三步，
>
> 第一是取指令；第二条，是判断rs和rt两个寄存器的内容是否相等，
>
> 那我们可以用一个减法来进行判断；第三步，是更新PC寄存器。
>
> 那么对于beq指令来说，所谓的分支，就是如何去改写PC寄存器，
>
> 所以它的重点在于第三步。我们先来看条件不成立的 情况，也就是else对应的PC=PC+4，
>
> 那在转移条件不成立的时候，我们是顺序执行后一条指令， 这就和其他的运算指令、访存指令是一样的。
>
> 那如果条件成立的时候，PC的更新条件则相对复杂一些，
>
> 其中也有PC+4，然后需要加上这个16位的立即数的符号扩展，并乘以4，

也就是说，在beq指令当中，所带的这个立即数， 也就是刚才在事例中出现的那个目标地址的标号True，

它实际的数值，是转移目标地址和下一条指令地址之间的差值，

而且这个差值是以4个字节，也就是32位为一个单位的， 那么这个规则，是在制度MIPS指令系统的时候约定的。

我们现在重点是看如何去实现相应的控制信号。

同样，我们直接来看第二步，那么对于这一步，

要做的操作包括，从寄存器堆当中，取出两个寄存器的内容，而且进行减法运算，

那这个操作和我们之前学习的减法指令需求是一样的，

因此，现有的结构不需要修改，就可以完成这个功能。

我们注意，当取回一条指令之后，rs的位域被连接到寄存器堆，

它所指定的寄存器的内容，会放到busA上，然后连接到ALU的一个输入端，

rt位域的信号会被连接到寄存器堆的Rb的输入端，

它所指定的寄存器的内容，会通过busB信号，再经过这个多选器

传递到ALU的另一个输入端，然后这个ALU就可以执行这个减法，

不过问题在于，之前的减法运算指令，会将这个ALU 运算的结果，通过这个多选器之后，写回到寄存器堆当中去，

而beq指令是不需要写回寄存器堆的，而且也不应该写回。

我们希望通过这个ALU，得出一个判断，就是这个减法操作的结果是不是0。

因此，我们还需要增加一个新的功能，来完成这样的一个判断，

判断一个数是否等于0，是非常简单的，所以我们可以很轻松地在ALU当中增加这个功能，

并让ALU提供一个信号的输出，标明当前的运算结果是否为0， 我们把这个信号命名为zero。

如果运算结果为0，ALU会把zero信号置为1，否则，置为0。

那因为运算结果是否为0，将会影响到IFU如何去更新PC寄存器，

所以我们需要把zero信号连接到IFU， 这样，我们就可以把第二步操作的描述补充完整。

那么在这一步操作中，这些控制信号又是如何设置的呢?

首先，下一个PC的选择方式，我们就不能再设置为加4了，

但是究竟如何更新，IFU还需要做一些工作，这个我们一会儿 再说。所以我们先把这个选择信号标记为branch，

然后我们再来看其他的控制信号。

现在我们已经知道，ALU要执行一个减法运算， 而且它的两个操作数都应该来自寄存器堆，

所以这一个多选器就应该选择通道0。

由此，扩展部件功能选择信号可以任意设置，

而ALU的功能选择信号，则需要设置为减法，那现在虽然我们新增了这个zero信号，

但是ALU原本的功能还是必须保持的。所以，当我们设置ALU执行减法运算时，它的输出

依然会是减法运算的结果，并送到数据存储器的地址端和下一个多选器的0号通道。

那为了保证数据存储器不被改写，那我们还要设置数据存储器的写使能信号为0。

那对于条件分支指令来说，它是不要回写寄存器堆的，

所以这个多选器无论选择哪一条通道，都是没有意义的， 那我们可以把它的选择信号任意设置为0或者1。

最后我们来看寄存器堆这一边，因为不需要回写寄存器堆，

所以我们必须要设置寄存器堆的写使能信号为0，以免错误地更改其中的内容，

那因为写使能信号已经设为0，那寄存器堆的写入寄存器的编号，则可以任意的设置。

这样，我们就可以看出，beq指令执行时真正有效的信号了。

不过我们要注意，这一步仅仅是完成了判断， 那我们还要根据判断的结果，对PC寄存器进行更新，

因此，这条指令的第三步和其他指令是不一样的。

那好，现在对于IFU来说，它有了两个输入的信号， 一个是之前就有的nPC_

select，还有一个是我们后来增加的zero， 而我们知道，对于IFU如何更新PC寄存器，

其关键，就是这个多选器如何选择的问题。

它的0号通道连接的是PC+4，1号通道连接的是分支指令的目标地址。

那现在我们这个选择信号应该如何生成呢？ 我们不妨把输入列一个表，来进行观察。

当nPC_select的信号为0的时候，就代表当前在执行的指令是运算指令，

或者是访存指令，而不是分支指令，那在这个时候，无论zero信号是0还是1，

这个多选器都应该选择0号通道，从而顺序地执行下一条指令。

而当nPC_select的信号等于1时，说明当前正在执行一条分支指令，

但如果此时zero信号为0，表示分支的判断条件不成立，

那这个多选器仍然应该选择0号通道，从而顺序地执行下一条指令。

只有当nPC_select的信号为1，说明当前是一条分支指令， 而且sero信号也为1，说明当前的判断条件成立，

这时，这个多选器才可以选择1号通道，从而将分支的目标地址更新到PC寄存器当中去，

这样在下一个时钟周期， 指令存储器就会将分支目标地址所指向的那条指令

的编码送出来，从而实现指令执行流向的改变。

那通过这张表，我们是否能够得出，这个多选器的控制信号的生成方法呢？

请你想一想。

给一点提示，实际上只需要一个逻辑门就可以了。

如果还没有想出来，那我们不妨回来再观察一下，对于

这个多选器的选择信号，只有在nPC_select和zero信号都为1时，它才会为1，

在其他时候，这个选择信号均为0。

那这个描述大家是不是很熟悉呢？这是哪个逻辑门的功能描述？ 我们还是来看最右边吧，

其实，只需要一个与门就可以了。

那基于这样的分析，我们就可以对IFU进行进一步的改造， 从而支持beq指令的需求。

不过这里还有一个问题，那就是分支目标地址究竟是如何生成的?

那现在我们就来完成最后这一项工作，这个分支目标地址有两个部分， 一部分是PC+4，

一部分是对立即数进行符号扩展，然后乘以4， 而这个立即数就是指令编码当中的低16位，

因此，我们先把这一部分信号连出来。

现在，我们把这个立即数取出来，连接到一个符号扩展的部件上，

对于这个符号扩展的部件，我们再增加一个很简单的小功能，就是向左再移动两位， 左移两位就相当于乘以4，

因此经过这个部件，我们将这16位的立即数扩展成了32位，并且完成了乘以4的操作。

那现在我们有了这个算式的后半部分，而前半部分是PC+4，

幸运的是，我们现在已经有了PC+4，就是这个加法器的输出，

那我们只需要直接把它连出来，然后再增加一个加法器，这样就可以得到了分支指令的目- 标地址，

现在这个IFU，我们也已经补充完整了。

现在，我们已经分析完了这个指令系统当中的最后一条指令了。

那我们已经知道，对于每一条指令， 每一个控制信号，我们应该赋予什么样的值。

但是还有一个问题，我们依然不是很清楚， 就是这些信号的值，是如何自动地产生的呢?

我们在下一节就来一起探讨这个问题。

506-控制信号的集成

自己设计出一个处理器是一件非常有意义的工作，

那现在，我们离这个工作的目标只差最后一点点了， 而最后的这一步，又是非常关键的，

那就是把我们之前所产生的这些控制信号集成起来， 并且，让这个处理器能够自动地工作。

现在我们已经分析完了这个事例的指令系统中所有的指令， 我们也知道了每一条指令应该设置怎么样的控制信号，

因此，我们现在就已经完成了这5个设计步骤当中的第4步，

接下来我们要做的就是考虑如何集成这些控制信号，形成完整的控制逻辑。

那我们先把之前的数据通路中的实现细节隐藏起来，就由一个方框

来表示，那控制这个数据通路正常运转的控制信号，一共有这样8个，

那这些控制信号是怎么产生的呢? 还是要从来自指令存储器的指令编码开始说起。

之前，我们已经从指令编码当中提取了若干的信号，作为数据通路的输入，

但我们只要回忆一下指令格式就可以知道，还有两个位域，我们没有提及，

这就是opcode和function，而正是这两个位域，是用来表明指令执行什么样- 的操作。

因此，我们正是需要用这两个信号，来产生下面这些控制信号。

实现这一部分功能的电路，就被称为控制逻辑， 也就是我们之前提到的控制器。要想知道控制逻辑

是怎么实现的，我们还是先来看一个例子。

我们以add指令为例，当我们取回一条指令的编码，如果是一条add指令，那它所需要

执行的操作就是这两个，一是将两个寄存器当中的数相加，并保存到另一个寄存器当中，

然后就将PC+4更新到PC寄存器当中。

对于这条指令所需要的各个控制信号的值，我们之前也已经详细分析过。

例如，这个RegDst信号应该为1，而RegWr信号也应该为1，

又比如，这个MemtoReg信号应该为0。

那我们把这些信号的值都摘出来，汇总成一张表格，

在这张表中，就是执行这条add指令时，各个控制信号的值，而add指令

对应的opcode域的值和function域的值，我们也列在上面。那用

同样的方法，我们也可以列出减法指令所对应的所有控制信号的值， 以及其他指令各自对应的控制信号的值。

这样，我们就得到了一张完整的表格，描述了每条指令和每个控制信号之间的关系。

那刚才我们是用一列一列的方式得到了这张表格，

然后我们换一个角度来看，如果按行来看，那我们就可以得到每一个控制信号的逻辑表达式，

我们就以第一行这个RegDst信号为例， 那从这一行可以看出，只有当前指令是加法或者减法指令时，

这个控制信号才需要为1，其他时候，都为0，就可以了。

那我们可以把这个控制信号的逻辑表达式写出来，

如果我们有一根信号标明现在是加法指令，即为add， 另而有根信号标明现在是减法指令，即为sub，

那这个RegDst的控制信号就等于add和sub这两个信号进行或操作。

那我们还得进一步分析，add和sub这两个信号应该如何生成。

那我们先来看add指令的编码，add指令是一条R型指令，

根据R型指令的位域划分，其opcode域都应该为0，而function域则指明- 了在R型

指令当中，这细分的指令类别。因此，对于I的指令来说，

就是opcode域为全0，而function域的第0位到第4位为0，

第5位为1。这样一种信号的组合就代表了当前指令是add指令， 那我们就可以把add指令的逻辑表达式写出来。

我们先用一根rtype信号，来表示当前是一个R型的

指令，然后我们来看后面的信号，就是function域的第5位， 对于add指令来说，这一位应该为1，

而function的第4位，一直到第0位，都应该为0， 所以我们将function

0到4，每一位都先取非， 然后与function域的第5位，以及这个rtype信号，一起进行与操作，

这样就可以得到了add信号。我们用类似的方法也可以得到sub这个信号。

那么rtype这个信号又是如何产生呢?

这非常地简单，因为R型指令就是opcode全为0， 所以我们只需要把opcode中的每一位都先取非，然后进行与操作，

那么当opcode全为0时，这个rtype信号就会为1。

我们把这些逻辑表达式连起来，就可以得到了RegDst这个控制信号的逻辑表达式，

有了这样的逻辑表达式，我们就可以用与门和非门画出产生这个控制信号的电路图来，

这样，对于这一个控制信号来说，它的控制逻辑就已经是确定的了。

那我们用同样的方法还可以得到其他控制信号的逻辑表达式，

例如ALUSrc这个信号，我们通过查询刚才的表格，可以知道，它是在

ori指令、load指令和store指令，这三条指令执行时是有效的。

因此我们注意下面，我们只要用类似的方法得到ori、load和store，

这三条指令对应的逻辑表达式，我们就可以得到ALUSrc这个控制信号的逻辑表达式了。

类似地，我们就可以得到所有控制信号的逻辑表达式。

那我们注意到，在下面这个区域，所有的信号都是来自指定的编码了，

而且，在对一些信号进行取非操作之后，这些逻辑表达式全都只有与操作，

而上面这个区域的表达式都是在下面这些信号的基础上，进行了或操作，

从而产生了最终的控制信号。因此，我们可以画出这样的一个控制器的示意图。

我们把指令编码中opcode的这六位和function的这六位 连出来，先接到一组由与门构成的逻辑电路中。

那么这一个逻辑电路就实现了上一页中，下面这片 区域的那些表达式，从而产生了一组中间信号，

那么再将这些信号接入一组由或门组成的逻辑电路，这组 逻辑电路就实现了上一页中上半部分那些逻辑表达式，

最后又可以得到了我们想要的所有的控制信号。

那么这些逻辑电路就是我们刚才所提到的控制逻辑， 这样我们就有了控制器的实现。

在这样的电路实现下，我们固定地将取回指令的最高六位和最低六位，连接到控制逻辑中，

那经过了这些与门和或门之后，就会自动地生成对应的控制信号，

而这些控制信号的值，肯定符合我们刚才对每条指令分析的需求，

这样在指令执行的过程中，就不需要任何智力因素的介入， 去分析当前到底是什么指令，以及需要产生什么样的控制信号，

一切都是通过这样的控制逻辑电路自动生成的， 这样，处理器就能一条接一条地取回指令，自动地执行下去。

那好，现在我们已经完成了控制信号的集成，并形成了完整的控制逻辑电路， 这最后一步工作，我们也就已经完成了。

现在，我们已经完成了一个处理器结构设计的步骤了，

虽然这个处理器只是单周期的，而且也只支持6条指令，

非常地简单，但是它确实是一个可以使用的处理器了。

如果你有兴趣，还可以继续深入到物理设计的环节，把它真正地制造出来。