

|  |
| --- |
| 32位MIPS单周期处理器设计 |

李晓畅 | 20307130261 | 2022年3月19日

# **一、简述**

微体系结构连接逻辑和体系结构，组合寄存器、ALU、有限状态机、存储器和其他逻辑模块，实现一种体系结构。本实验设计的32位MIPS单周期处理器就处于这一层次。这意味着我们需要在熟悉算数单元与存储单元这些更低抽象层次设计的基础上进一步理解寄存器、指令、存储器等概念以实现更高的抽象。

本实验设计的MIPS处理器产生于上世纪八十年代，它经典、简洁，虽然相对简单，但正如MIPS是最早出现的商业RISC架构那样，学习设计MIPS将有助于我们理解真实的商业体系结构。

# **二、设计原理**

**１. 设计原则**

Patterson和Hennessy提出了MIPS 体系设计的四个准则，即：

1） 简单设计有助于规整化

2） 加快常见功能

3） 越小的设计越快

4） 好的设计需要好的折中方案

我们将在下面的具体实现中深刻体会到这几点。

**２. 体系结构状态与指令集**

计算机的体系结构包括指令集和体系结构状态。

一台计算机能执行的机器指令的集合称为该机的指令集或指令系统，它是构成程序的基本元素，也是硬件设计的依据。因而指令系统的设计设计是计算机系统设计的关键之一。MIPS采用RISC技术，指令系统简单，控制逻辑得以大大简化。

MIPS处理器的体系结构状态包括程序计数器和32个寄存器。这些原件是任何MIPS处理器的必要组成部分。基于当前体系结构状态，处理器执行一条具有特定数据集的特定指令，将产生一个新的体系结构状态。

为了使得微体系结构易于理解，我们将首先着手实现MIPS指令系统的一个子集，进而考虑将其加以扩展，以实现更复杂的功能。指令选择将在下面阐述。

**3. 时钟周期**

单周期CPU指的是一条指令的执行在一个时钟周期内完成,然后开始下一条指令的执行,即一条指令用一个时钟周期完成。电平从低到高变化的瞬间称为时钟上升沿,两个相邻时钟上升沿之间的时间间隔称为一个时钟周期。时钟周期一般也称振荡周期

CPU在处理指令时,一般需要经过以下几个步骤：

1） 取指令：根据程序计数器PC中的指令地址,从存储器中取出一条指令,同时,PC根据指令字长度自动递增产生下一条指令所需要的指令地址,但遇到"地址转移"指令时,则控制器把"转移地址"送入PC,当然得到的"地址"需要做些变换才送入PC。

2） 指令译码：对取指令操作中得到的指令进行分析并译码,确定这条指令需要完成的操作,从而产生相应的操作控制信号,用于驱动执行状态中的各种操作。

3） 指令执行：根据指令译码得到的操作控制信号,具体地执行指令动作,然后转移到结果写回状态。

4） 存储器访问：所有需要访问存储器的操作都将在这个步骤中执行,该步骤给出存储器的数据地址,把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

5） 结果写回：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

因而对于单周期CPU,就是在一个时钟周期内完成这五个阶段的处理。

具体到本次的设计，我们选择100MHZ时钟作为系统时钟，又选择其下降沿为IO时钟。

# **三、实验方案**

**1. 实验目的**

1）设计实现32位MIPS单周期处理器

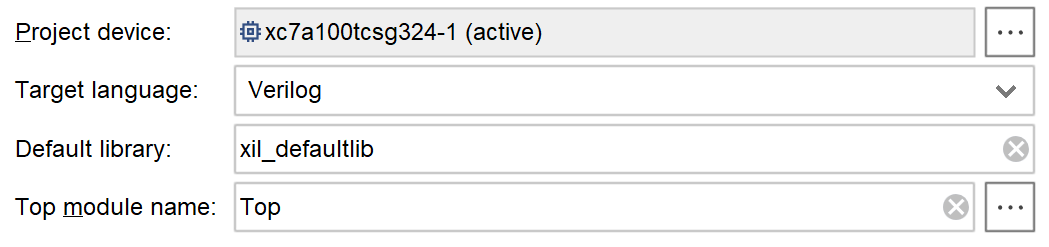
2）实现指令扩展

3）实现IO接口

4）加深对微体系结构的理解

**２. 环境和器材**

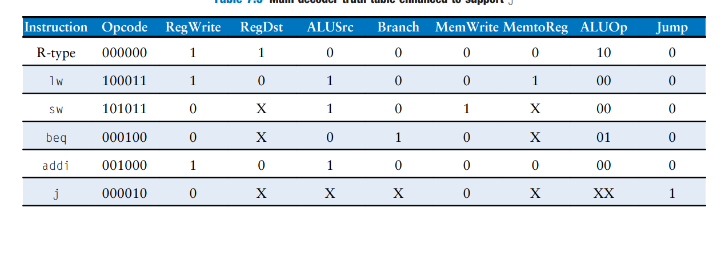
使用Ｖｉｖａｄｏ２０１８．３设计开发，应用开发板测试验证。具体如下。



**３. 设计过程**

正如前面提到的那样，我们将考虑首先实现MIPS指令集的一个子集，接着再加以扩展。为了保证能够实现需要的功能，我们选择了下面几条指令。

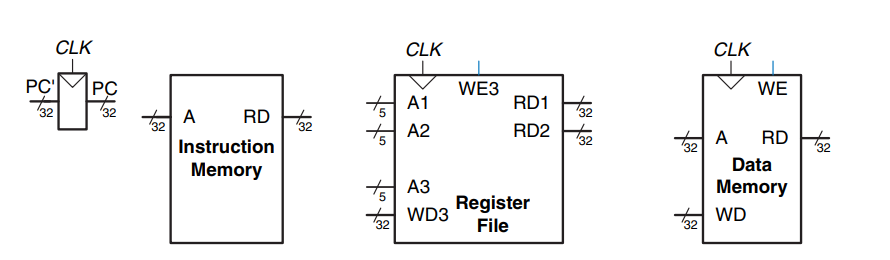
前面五条我们将在设计时直接实现，而最后一条指令和立即数逻辑运算将以扩展的形式实现。



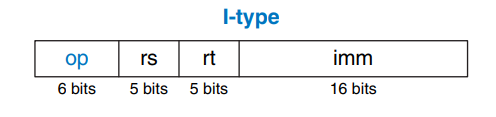
确定了需要实现的指令集，再加上一些必要的模块，我们接下来就可以考虑数据通路的设计和控制逻辑的设计了。

设计复杂系统时，一种好方法是从包含状态原件的硬件开始，而这些原件中就包括存储器和体系结构状态。然后在这些存储组件之间增加组合逻辑基于当前状态计算新状态。从部分存储器中读指令，然后装入和存储指令从一部分存储器读或写数据。

基于这一设计思想，在分析指令功能及所需要的组件后，我们可以从下面这些原件开始考虑。

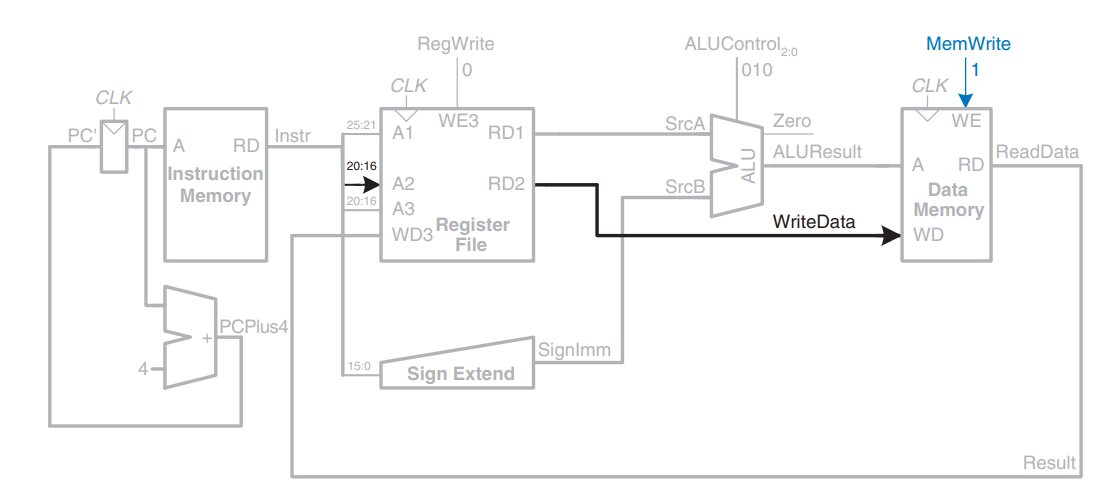


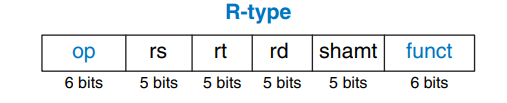
首先我们可以着手实现，指令构造数据路径，然后考虑泛化该数据路径以实现更多指令。

lw、指令就是一条I类型指令，有着这样的结构。

具体来说，op字段决定指令指令的操作。字段具体代表源操作数或目的操作数由具体指令决定。对于指令，rt是目的寄存器，而和是源操作数。

可以看出，立即数位数只有十六位，所以需要扩展，也就是要求添加扩展原件。内存地址将由一个基地址和一个变地址共同决定，因而需要一个原件实现运算，也就是需要算数逻辑单元ALU。另外，在执行完一条指令后，我们希望继续执行下一条指令，所以需要改变PC的值，具体到３２位处理器，我们要将PC加４。添加上述的原件，并添加适当的控制信号，我们就可以实现指令和指令了，也就是下面这样。



然后是R类型的指令实现。一条32位的R类型指令是这样的。

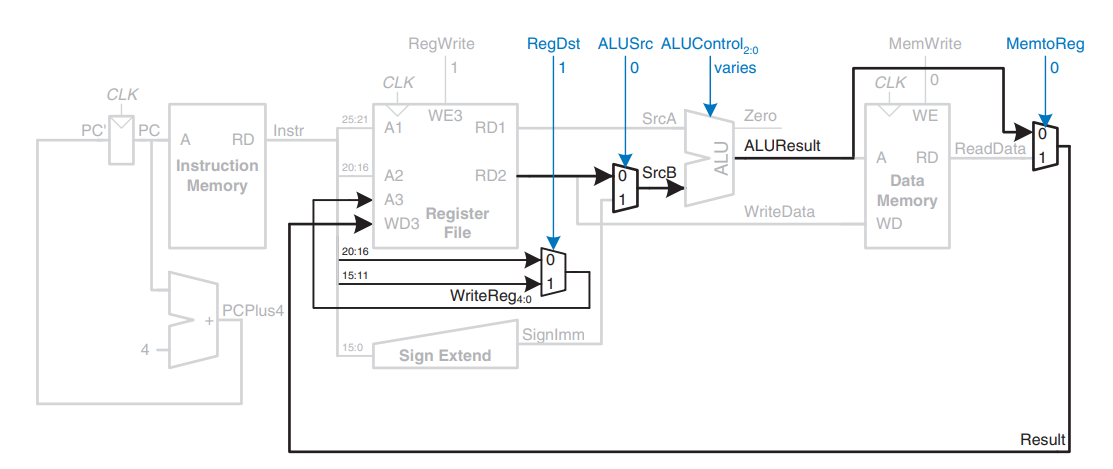
具体来说，指令操作码包括：

指令操作编码为op和两个字段各六位，其中对R类型指令而言op总为0，而决定R操作的类型。

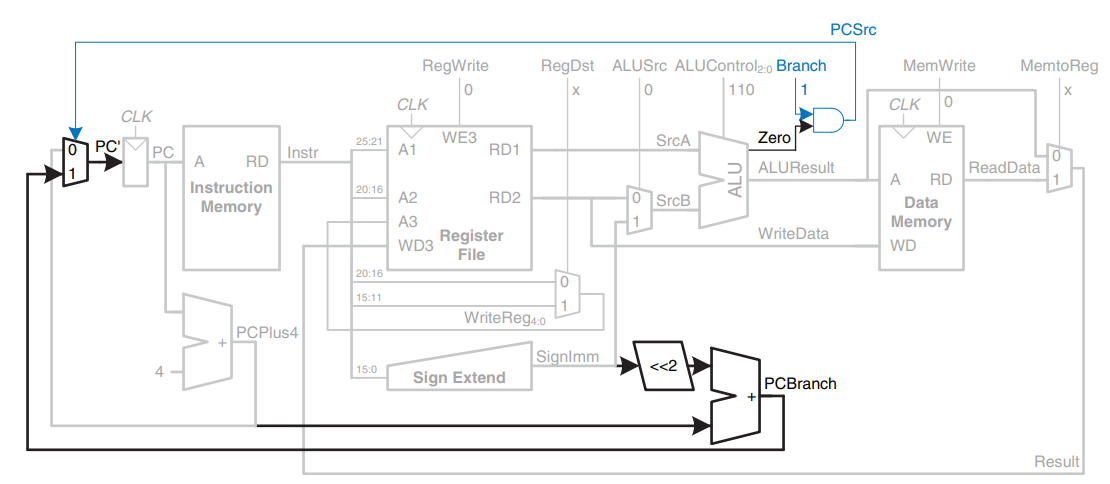
指令操作数编码为三个字段，即。前两个作为源寄存器，而最后一个是目的寄存器。

仅用于移位操作，以五位二进制数表示移位位数，而在执行其他指令时为0。

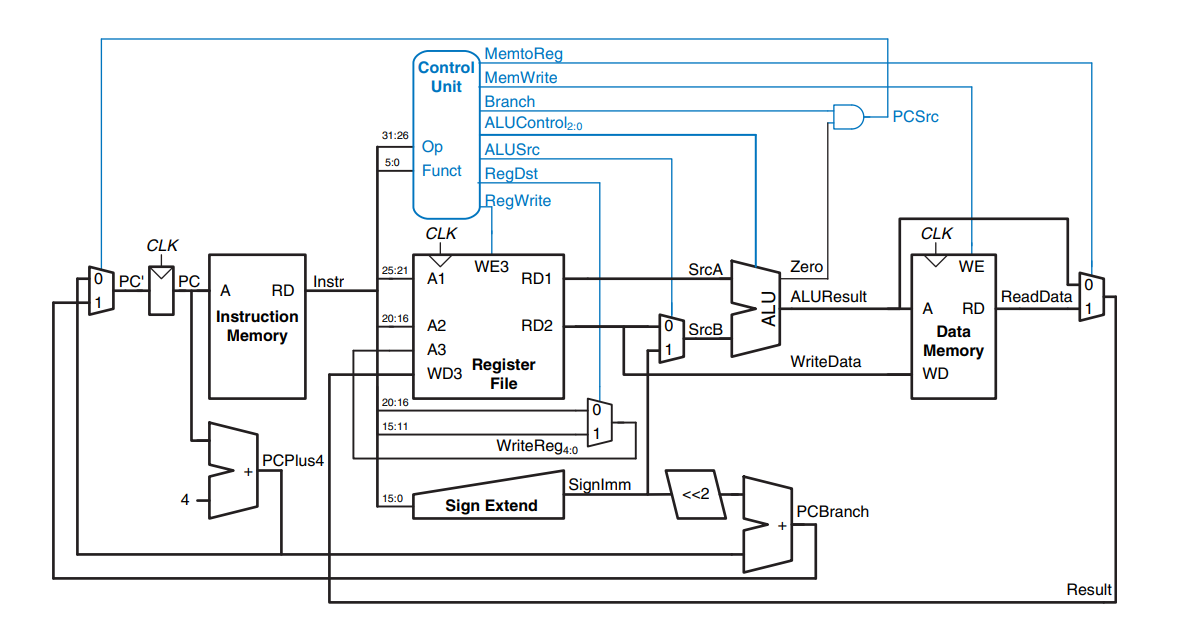
考察与之前设计电路的差别，我们发现精妙的设计使得很多通路都是相同的，只有很少的一些路线有所变化。因而我们可以添加多路器和控制信号就可以实现R类型的指令了。



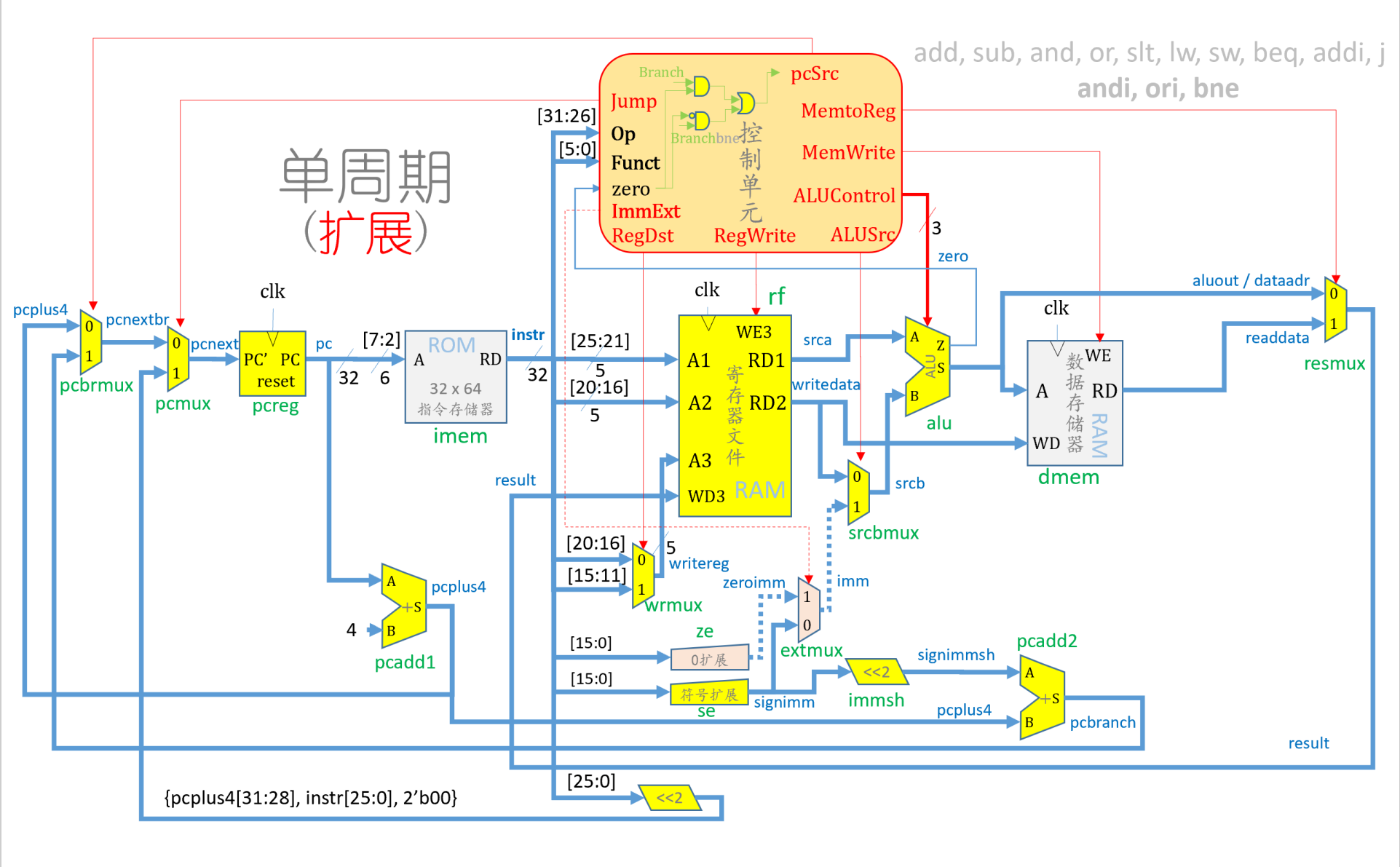
类似的，我们继续添加组合逻辑以实现指令，于是就得到了下面这样的结构。



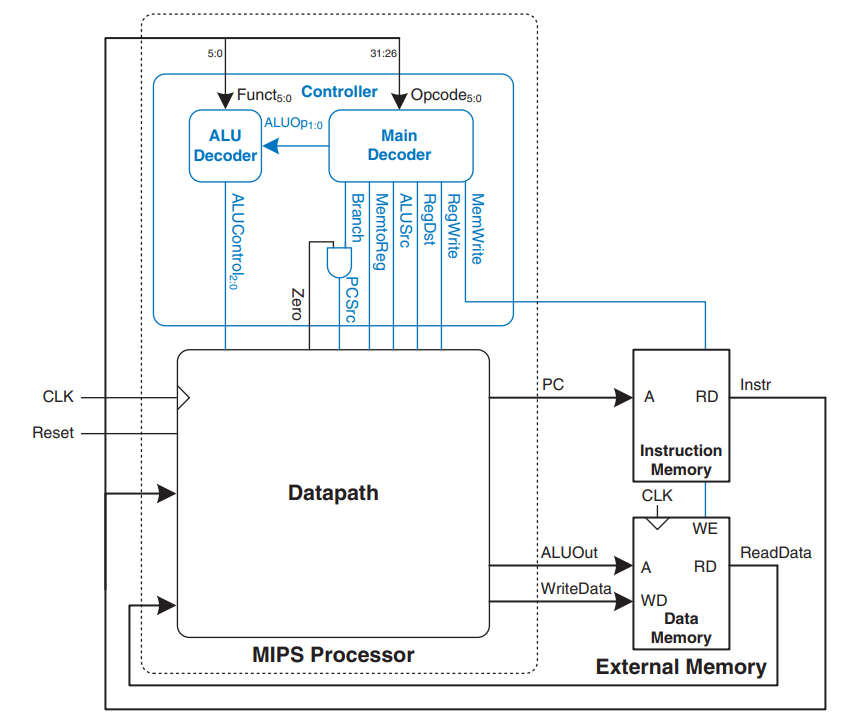
最后，我们需要一个根据指令类型产生控制信号的模块，也就是具体来说就是主译码器模块和ALU译码器模块。参照我们在设计指令时的设计，我们不难设计出这些模块。下面我们将它们同已有部件组合起来，就得到了下面这样的设计。



这已经实现了前几条指令了。接下来我们继续考虑跳转和立即数逻辑运算。相较于算数运算，逻辑运算中的立即数不应该实行符号扩展而应该实行０扩展。所以我们可以添加０扩展模块及多路器，并设计控制信号实现。对于无条件跳转指令，它需要直接计算出下一条指令的地址。考虑ｊ指令的２６位立即数，我们不难发现只需要将其左移两位作为下一地址即可。添加对应原件、多路器和控制信号，就可以得到下面这样的设计。



最后，为了实现同外界交互，我们还要增加IO接口。这里我们将实现存储器映像IO接口。在该方法中，I/O接口与存储器共用同一个地址空间；每一个I/O设备占用存储器空间的一个地址；CPU利用等指令对I/O设备的管理； CPU利用存储器读写信号对I/O设备进行读写控制。换言之我们需要做的就是建立IO与存储器的通路，并设计合适的控制信号即可。那么就可以得到下面这样的设计了。

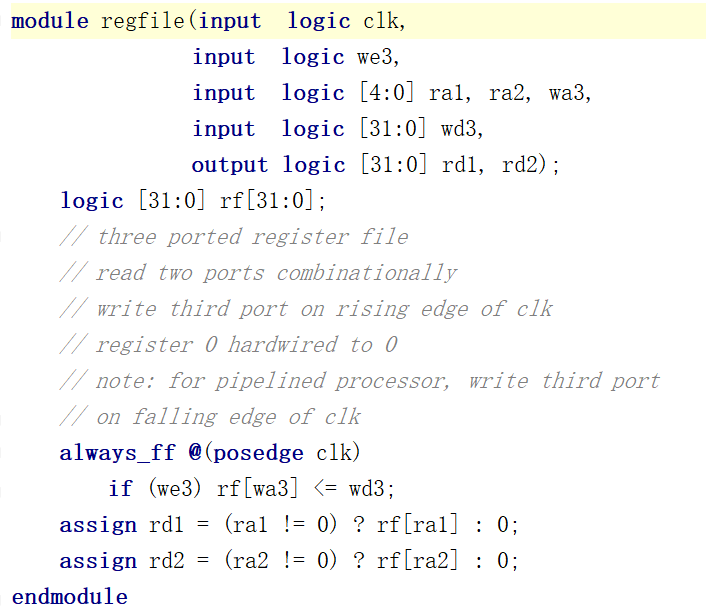


# **四、关键代码、仿真与开发板照片**

**１. 存储器和体系结构状态**

正如我们在设计过程中提到的那样，存储器和体系结构状态在设计中有着重要的地位。我们将首先实现它们。

比如下面就是一个寄存器文件模块。它包含一个时钟信号，两个读地址及其输出，一个写地址及其输入，以及一个控制信号。在时钟的上升沿，如果控制信号允许，将对应寄存器进行写操作；另外，将根据读地址取出两个寄存器中的值。

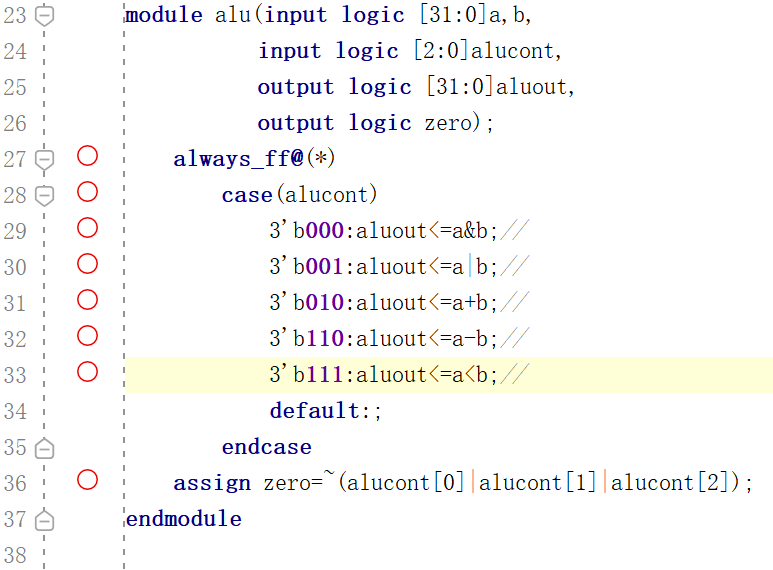


指令存储器，数据存储器和PC实现与之类似，这里就省略了。

**２. 主要模块**

１） ALU模块

ALU模块在处理器中有着重要作用，对运算有着巨大影响。我们可以像下面这样设计它。包含两个３２位输入，一个三位控制信号决定计算类型，一个零信号作为标志位输出，以及一个３２位输出信号。只需要根据控制信号选择正确的运算并加以输出即可。

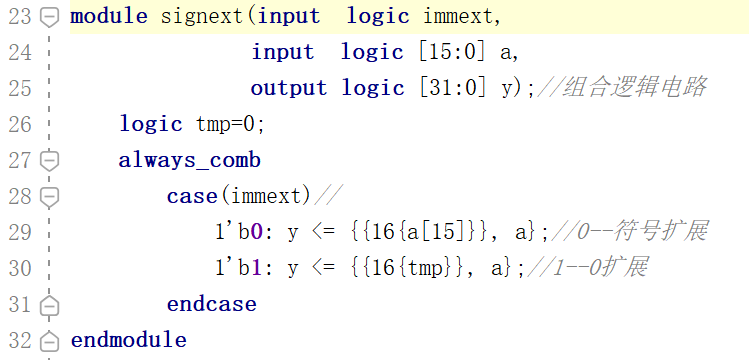


加４模块和移位模块与之类似，这里就不赘述了。

２） 符号扩展和０扩展模块

我们的３２位处理器经常需要接受一些小于３２位的信号，为此我们就需要对它们进行扩展。而扩展又分为符号扩展和０扩展。因而我们可以这样设计扩展模块。

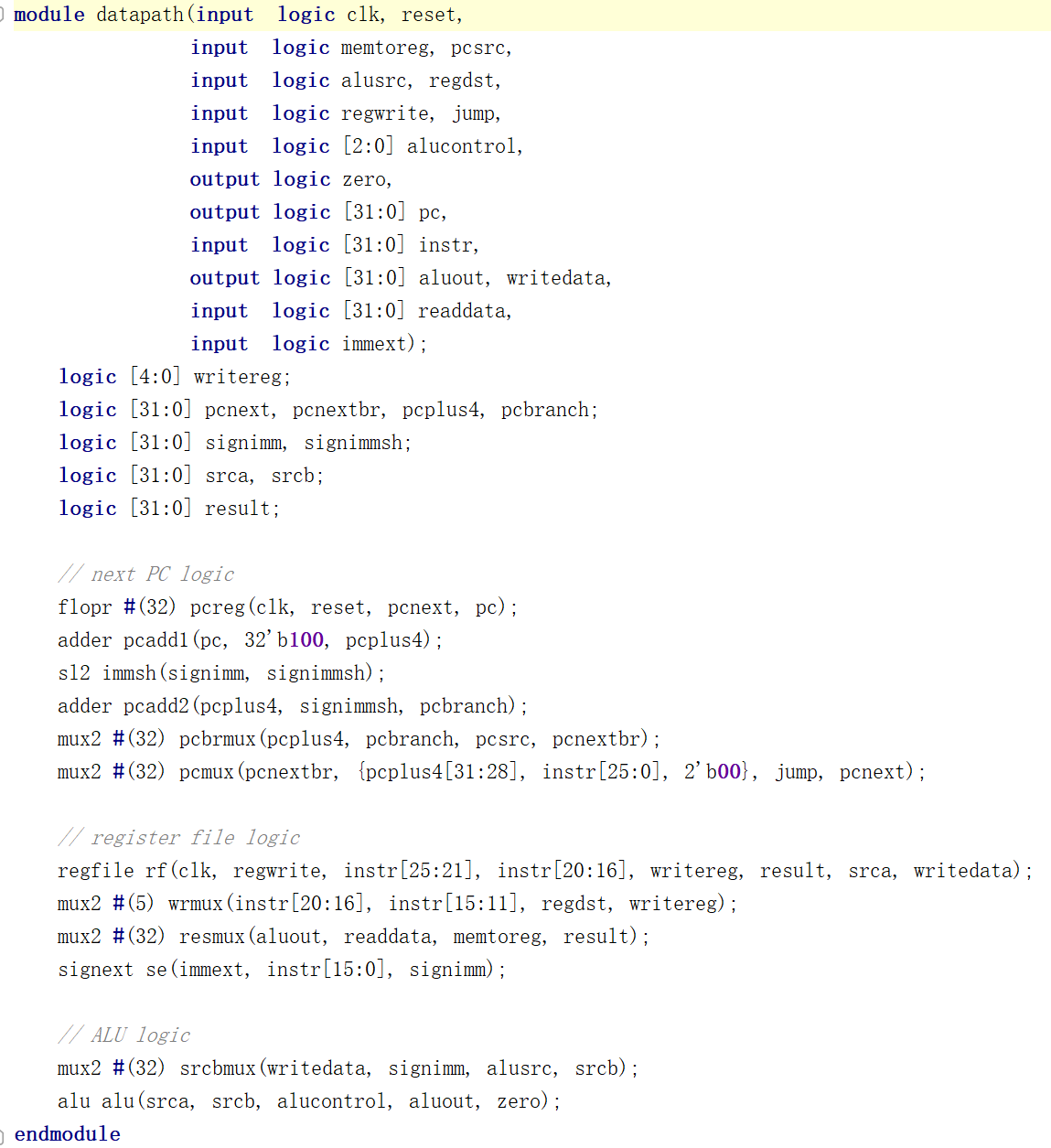
它包含一个控制信号决定进行０扩展还是符号扩展，一个１６位输入和一个３２位输出。这里的设计中整合了两种扩展模块。



**３. 数据通路**

就像在设计过程中提到的那样，有了上面提到的模块，我们就可以设计数据通路了。

数据通路例化了前面提到的原件，将之有机连接以实现指令。我们将数据通路作为模块单独封装了起来，并提供了同其他模块，比如控制模块间的接口，这样我们在日后更改的时候这一模块仍旧是独立的。具体来说就是下面这样的实现。

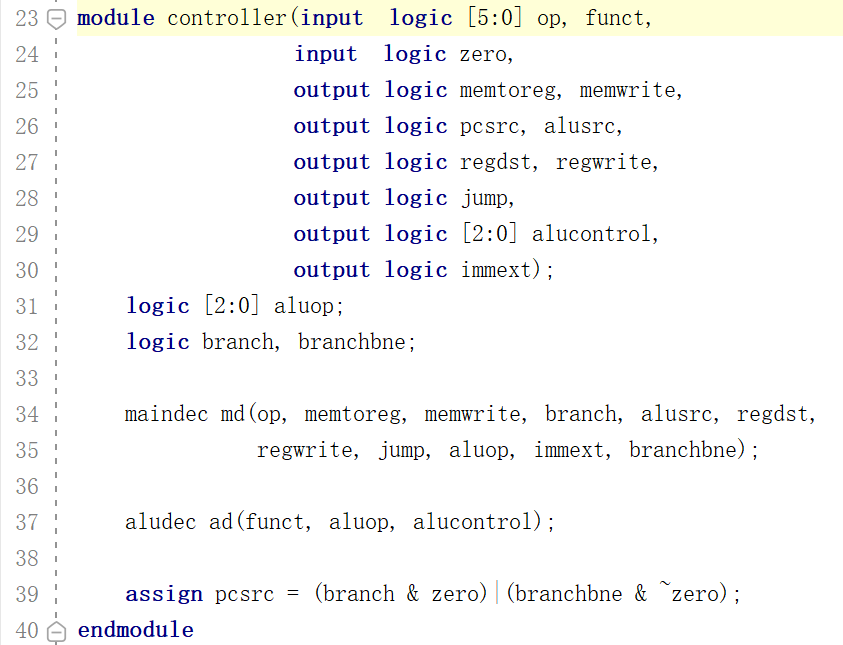


**４． 控制模块**

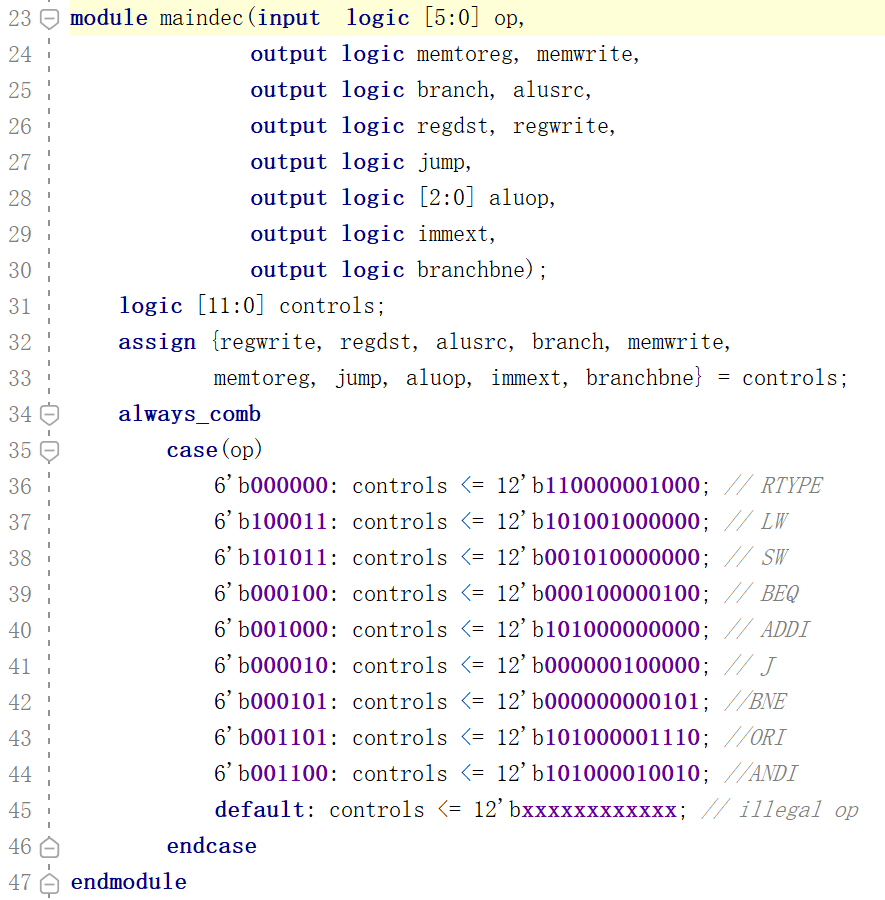
为了根据和字段计算控制信号我们需要设计一个控制单元。

我们将单独设计主译码器与ALU译码器，再设计一个模块来例化它们。这样我们在扩展和更改时将大大获益。

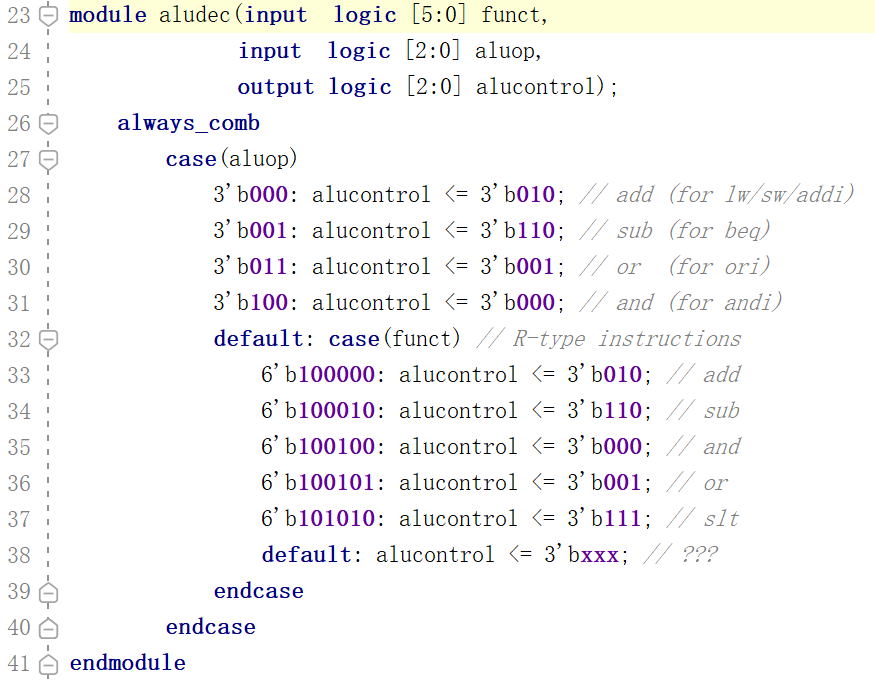
控制模块包含了输入的控制字段和标志，在处理后输出控制信号。



主译码器模块基于字段产生控制信号，并输出。

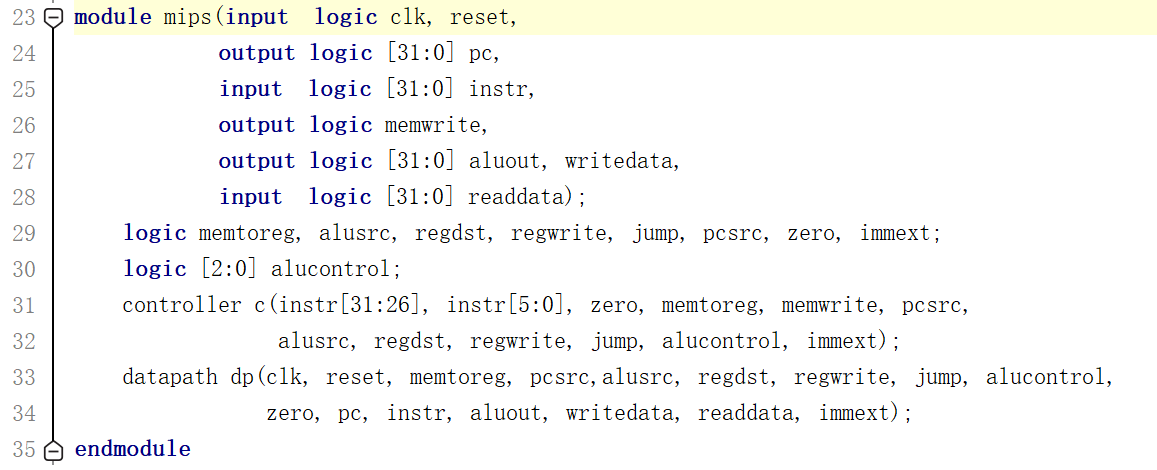


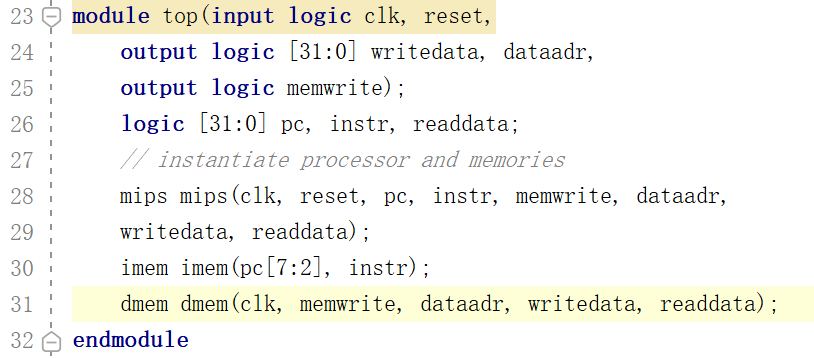
而ALU译码器根据主译码器信号和字段产生控制ALU的信号。



**５． 顶层模块**

在完成了上面这些设计后，我们只需要设计一个模块包含数据通路和控制模块，再设计一个顶层模块包含模块和指令存储器模块就可以了。比如下面这样。



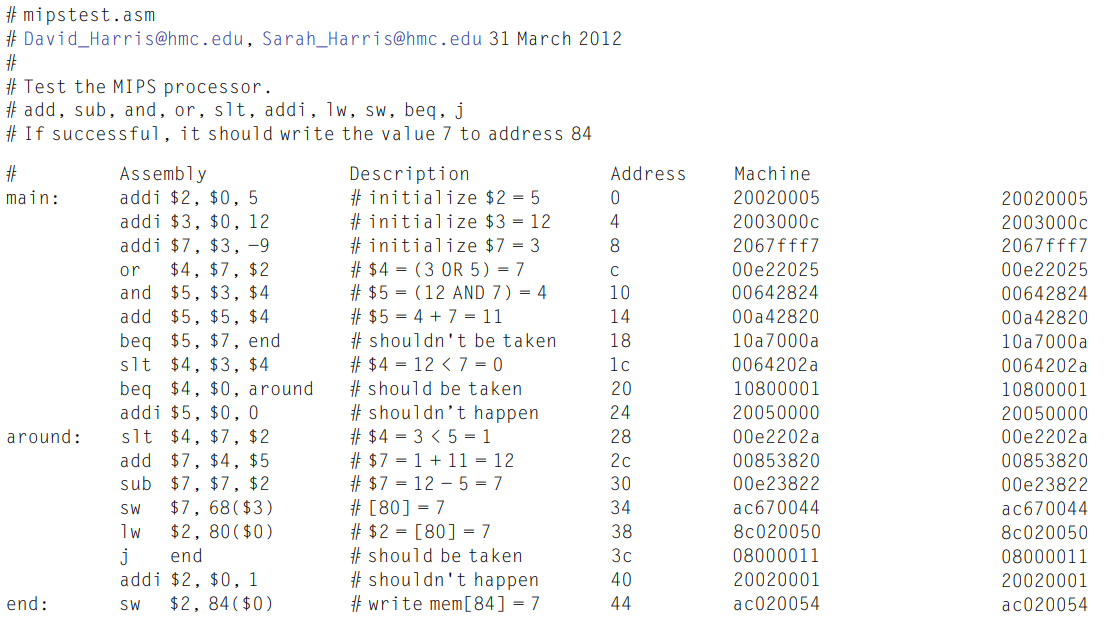


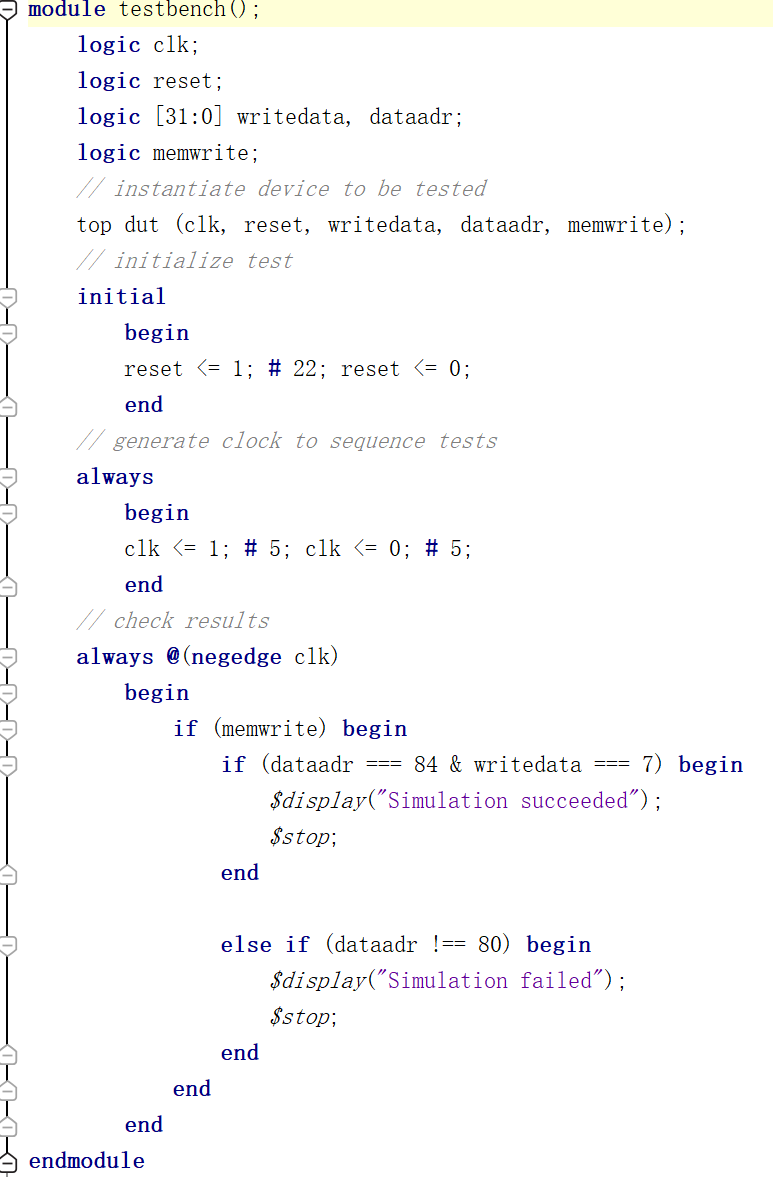
**６. 仿真**

在完成设计后我们就可以设计仿真了。为了验证指令可以正确执行，我们可以在指令存储器里写一段程序加以测试。

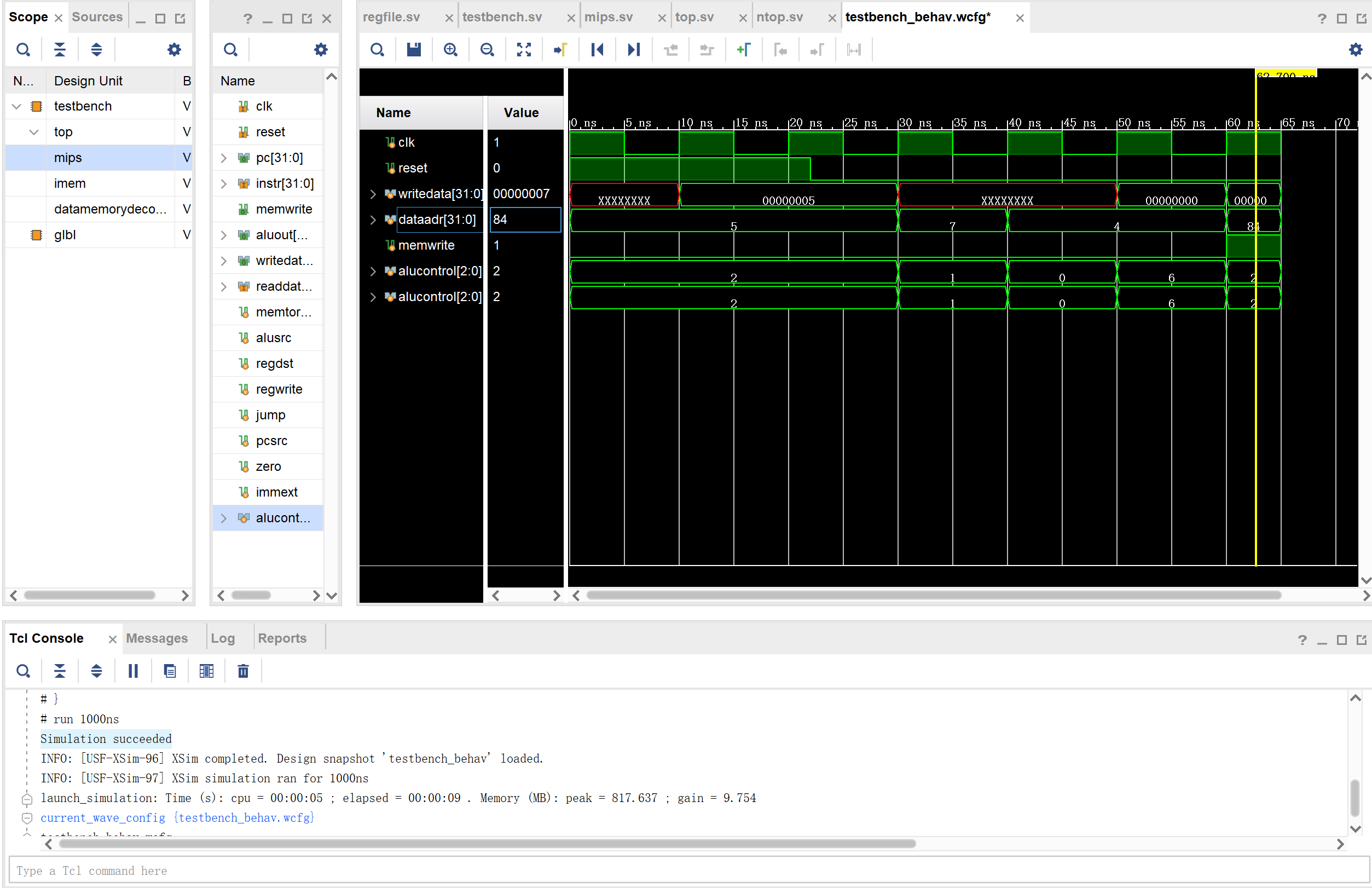
1. 未扩展的处理器

这样一段代码配合测试程序可以测试前几条指令的功能。



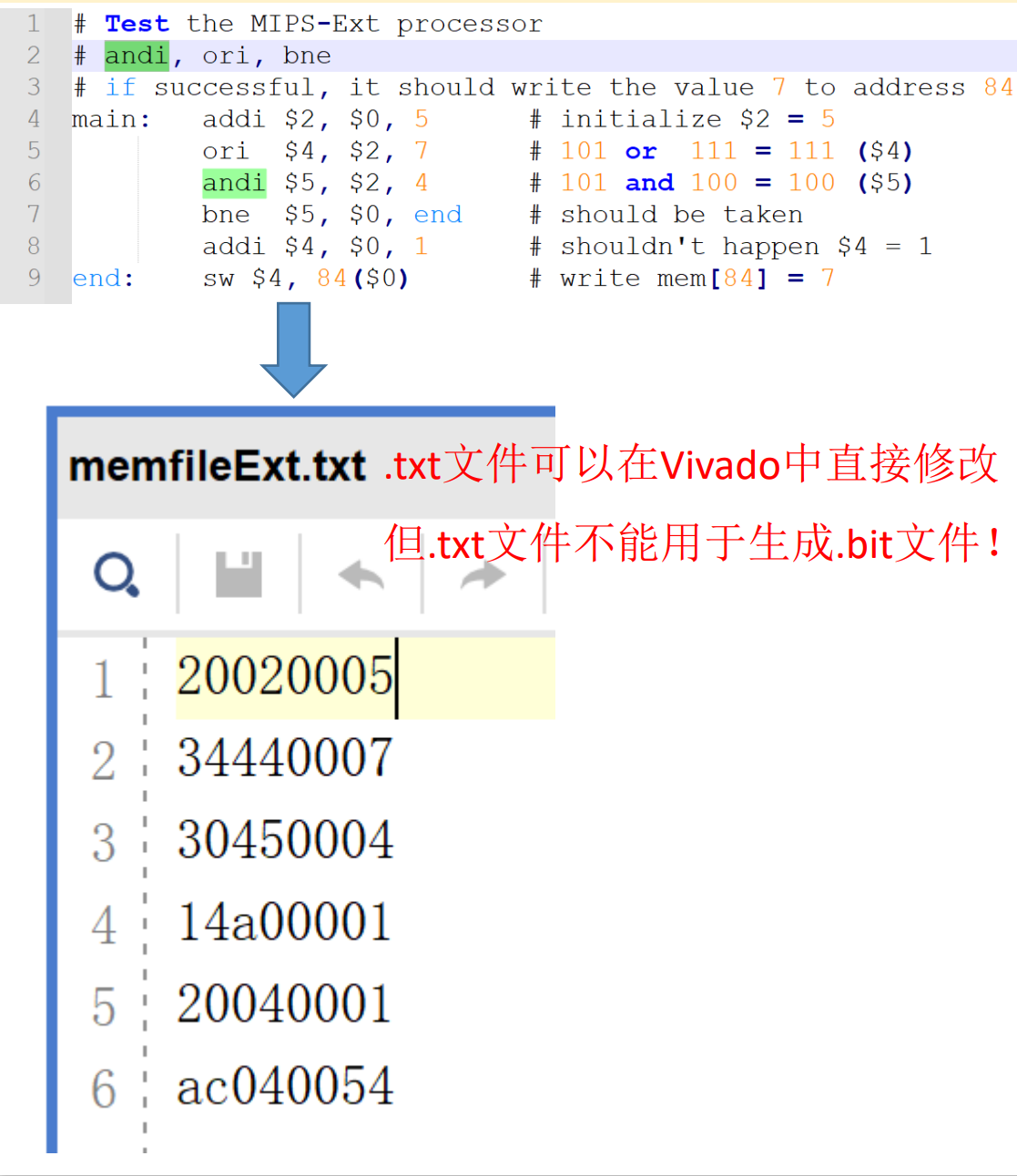


于是就可以得到这样的仿真结果。仿真成功。

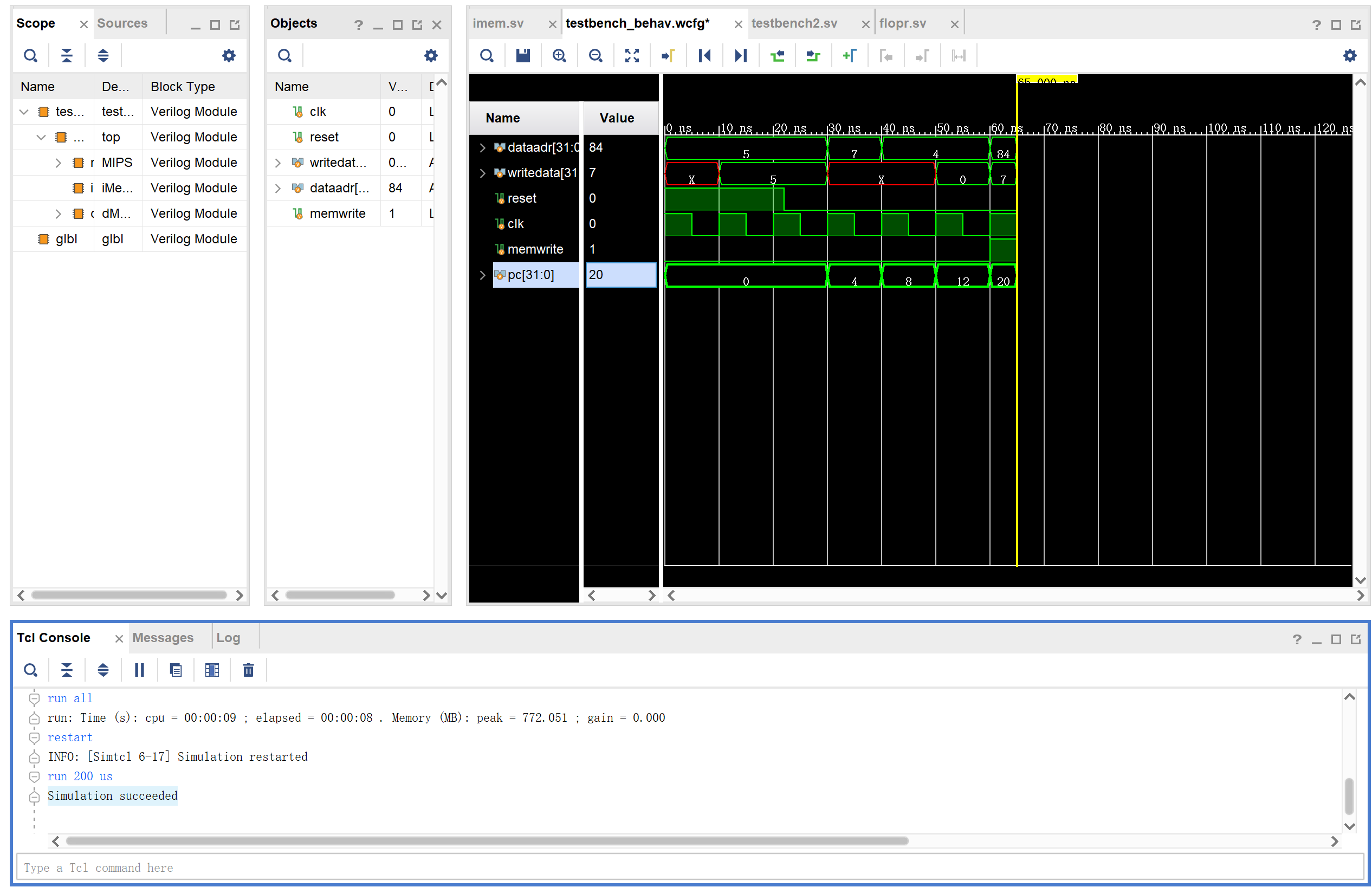


1. 扩展

在设计过程中我们就已经加入了ｊ指令和ａｄｄｉ指令的控制信号，我们的扩展模块也支持０扩展。所以我们只需要另外设计程序就可以了。

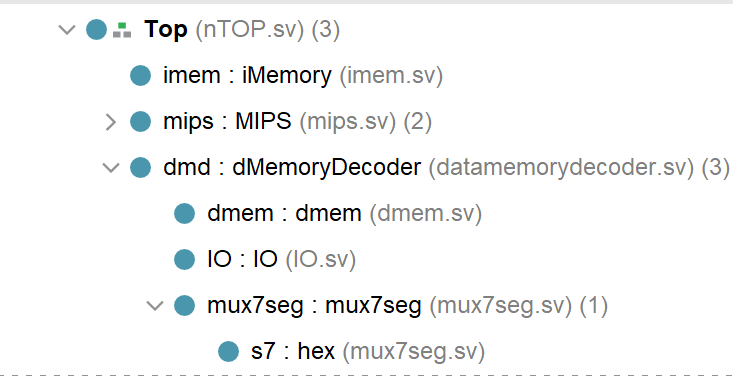


同样可以得到预期结果。

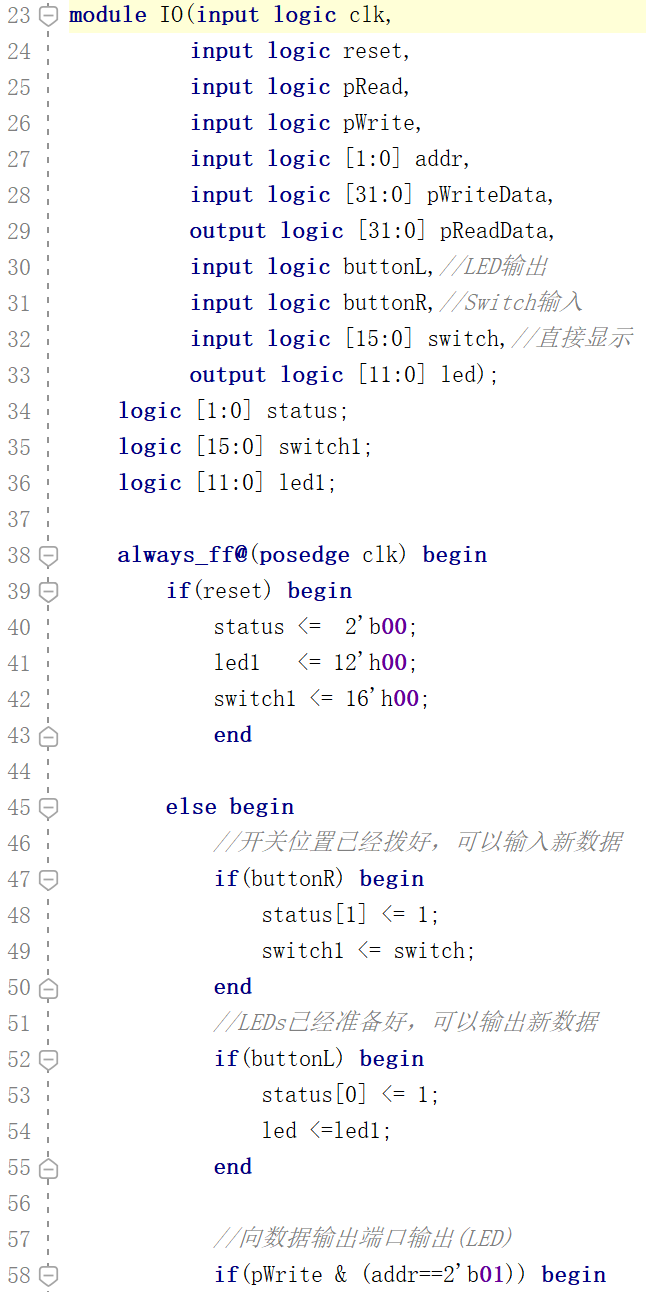


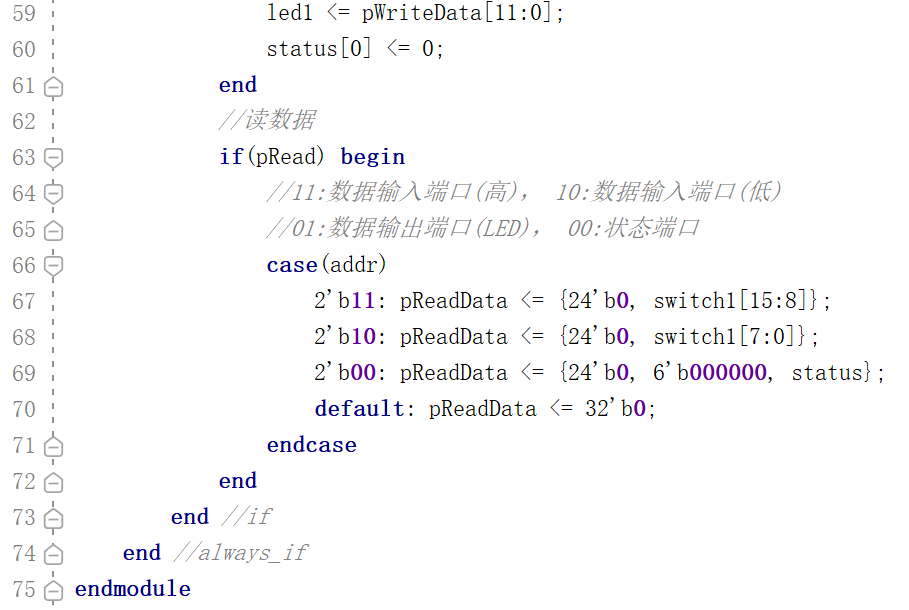
**７. IO接口设计**

由于我们之前设计已经实现了大多数模块，下面我们只需要设计一个新模块，并设计新的顶层模块就可以了。



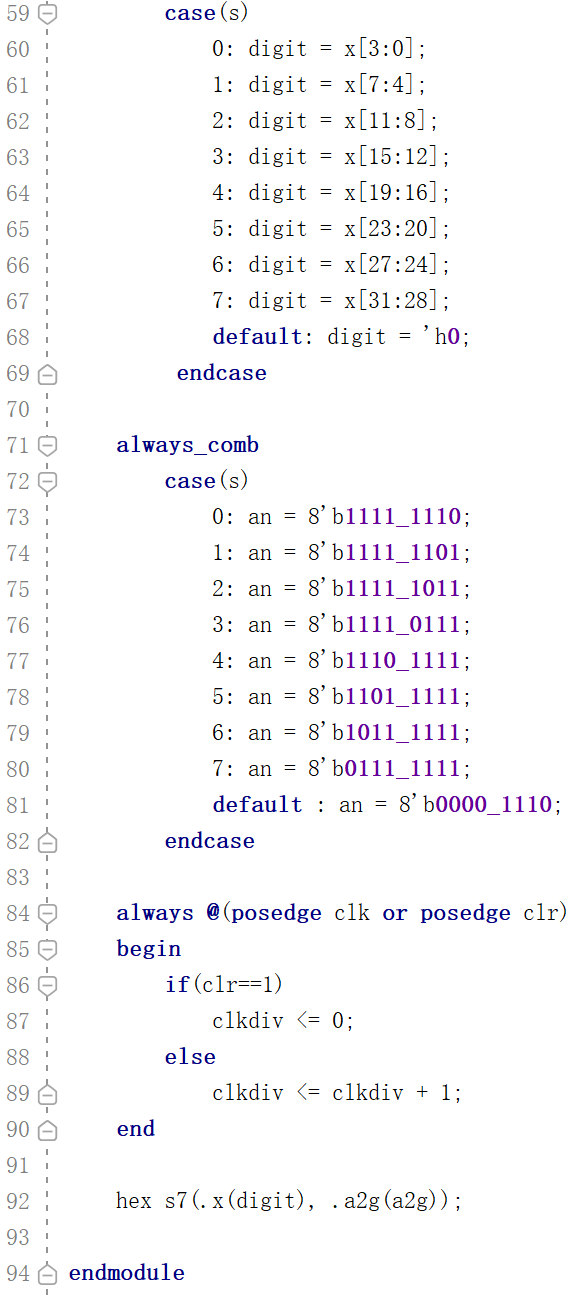
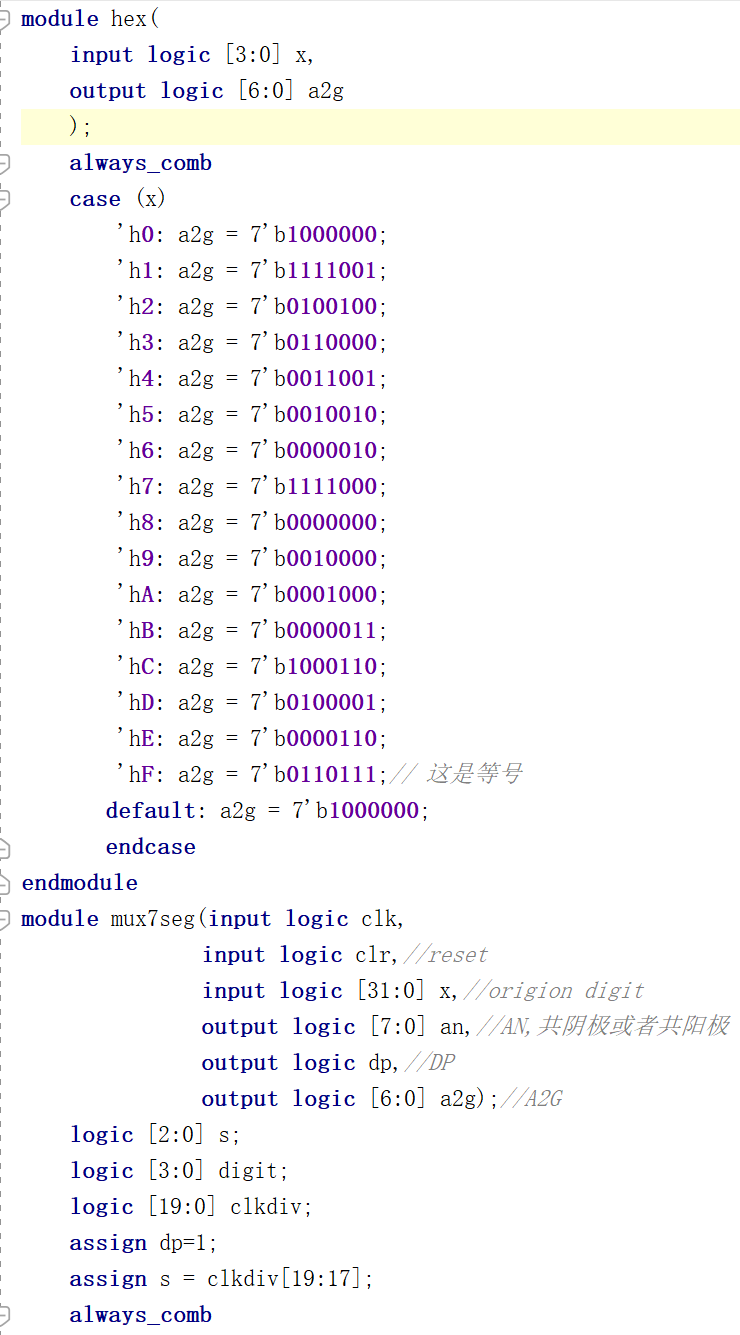
在原来数据存储器上加上附加的模块，使之能够支持IO操作。具体来说，我们需要一个额外的IO模块、一个七段数字管模块，以及一个控制模块。下面是IO模块。





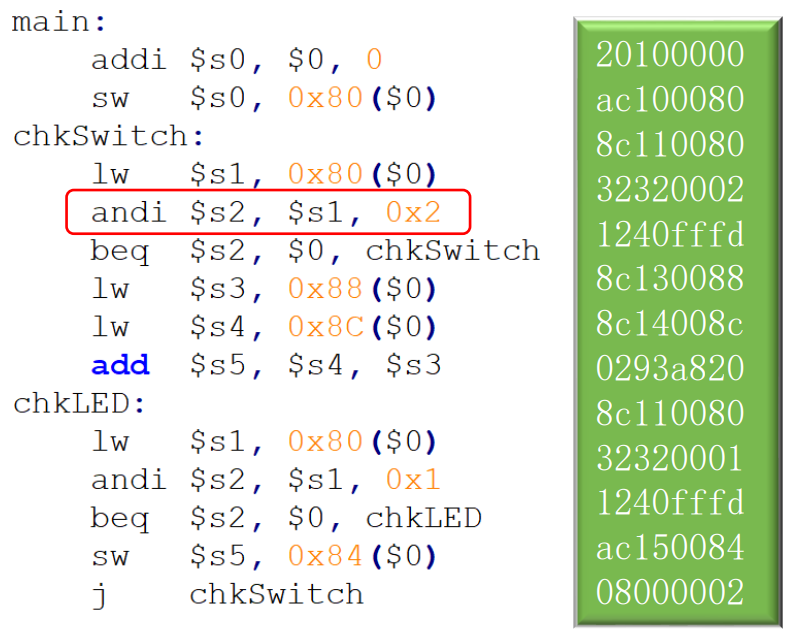
它可以从外界读取数据将之输出，也可以输出数据。

而七段数字管模块可以让输出结果显示在LED灯上。



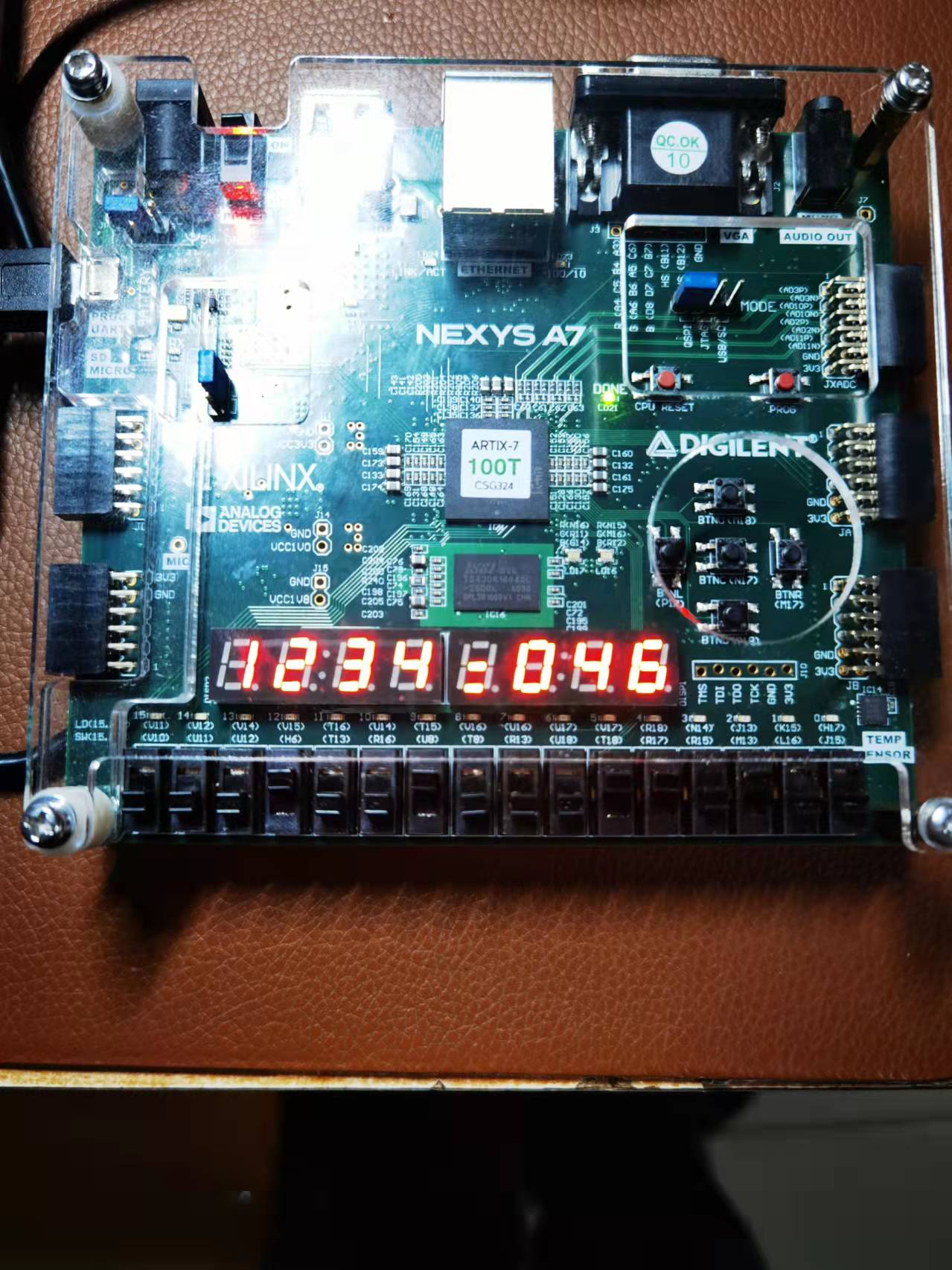
再加上控制模块，我们就实现了IO接口的设计。

在程序存储器中写入下面的程序。



生成二进制文件，写入开发板。

单击中间按钮(系统清零)，再单击左按钮(LED输出)，则清零； 单击右侧按钮(开关输入)，再单击左按钮(LED输出) ，显示相加结果。最终效果如下。



# **五、思考与收获**

**１. 精巧的指令集**

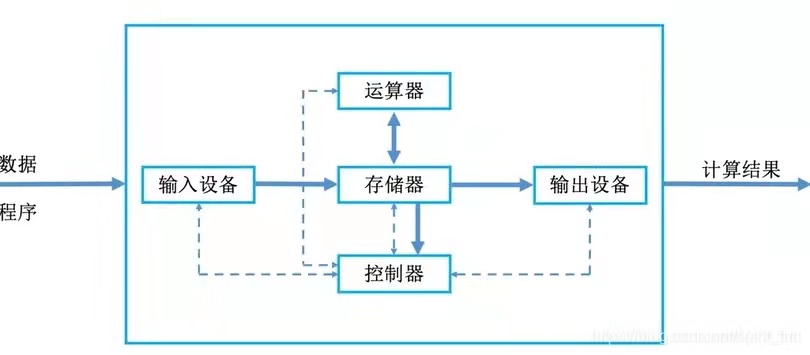
指令系统的设计设计是计算机系统设计的关键之一。从本次的设计中可以发现，一个精巧的指令集可以让设计变得更加简单而高效。

考察我们的R型指令。它有着两个控制字段，但是它们分别处于指令的最高六位和最低六位，而非连续的十二位。这一设计似乎反常，但却是很有意义的。因为我们还要实现其他类型的指令，而这些类型的指令只有前六位是控制字段，接下来的五位在很多指令中都是作为操作数的寄存器，所以R型指令也按照六位控制字段加寄存器的模式设计可以免去对寄存器地址的复用。另外，虽然我们可以把所有指令都使用统一长度的控制字段，但这样会使得有些指令位数不够，而有些又位数太多。由于R型指令操作数只有三个五位的寄存器，所以我们为它选用了共十二位的控制字段和五位移位字段，这充分利用了其长度，又为其他指令留下空间。

类似的，ｓｗ指令并没有把目的操作数放在第一个，而是采取类似ｌｗ指令的方式。我们可以设想，如果ｓｗ指令依照惯例将目的操作数写在前面，我们将不得不加上很多电路和多路器，需要产生更多控制信号。而由于这一反常设计，这两个操作所使用的电路在很多地方都是相同，电路设计得以简化。

这样的例子还有很多，而这些精巧的设计无疑让我们的设计更简单、更精巧，这不仅可以加快速度，还可以减少不必要的错误。

**２. 以存储器为核心**

在设计中，我们采取的策略是从包含状态原件的硬件开始进行设计的模式。这一模式同以存储器为核心的思想不谋而合。

**３. 收获**

在本次实验中，我们设计的处理器有着鲜明的结构层次。从更低的抽象等级向更高的抽象等级看，可以更好的理解原件间的相互作用，协同工作；从更高的抽象层次向更低的抽象层次看，可以了解自向下的设计思想，了解体系结构的精妙之处。