**中山大学数据科学与计算机学院本科生实验报告**

**（2019学年秋季学期）**

课程名称：**计算机组成原理实验**  任课教师： 郭雪梅 助教：丁文、汪庭葳

|  |  |  |  |
| --- | --- | --- | --- |
| 年级&班级 | **2019级04班** | 专业(方向) | **计算机科学与技术（超算方向）** |
| 学号 | **19335112** | 姓名 | **李钰** |
| 电话 | **19847352856** | Email | **1643589912@qq.com** |
| 开始日期 | **2020年11月20日** | 完成日期 | **2020年11月27日** |

**一、实验题目**

单时钟周期 CPU 的设计实验

1. **实验目的**
2. 理解MIPS常用的指令系统并掌握单周期CPU的工作原理与逻辑功能实现。
3. 通过对单周期CPU的运行状况进行观察和分析，进一步加深理解。
4. **实验内容**
5. **实验原理**

**单时钟周期CPU**

单周期 CPU 的特点是每条指令的执行只需要一个时钟周期，一条指令执行完再执行下一条指令。再这一个周期中，完成更新地址，取指，解码，执行，内存操作以及寄存器操作。由于每个时钟上升沿时更新地址，因此要在上升沿到来之前完成所有运算，而这所有的运算除可以利用一个下降沿外，只能通过组合逻辑解决。这给寄存器和存储器RAM的制作带来了些许难度。且因为每个时钟周期的时间长短必须统一，因此在确定时钟周期的时间长度时，要依照最长延迟的指令时间来定，这也限制了它的执行效率。

单周期CPU在每个CLK上升沿时更新PC，并读取新的指令。此指令无论执行时间长短，都必须在下一个上升沿到来之前完成。

CPU 的顶层结构实现。主要器件有程序计数器PC、程序存储器、寄存器堆、ALU、数据存储器和控制部件等。所有的控制信号简单地说明如下：

其中，控制单元(Ctrl Unit)定义如下：

（1）JUMP：为1时，选择跳转目标地址；为 0时，选择由Branch选出的地址；

（2）MemToReg：为1时，选择存储器数据；为0时，选择ALU 输出的数据；

（3）Branch：为1时，选择转移目标地址；为0时，选择PC +4（图中的 NextPC）；

（4）MemWrite：为1时写入存储器。存储器地址由ALU的输出决定，写入数据为寄存器rt的内容；

（5）ALUOP：ALU控制码；

（6）ALUSrc：ALU操作数B的选择，为1时，选择扩展的立即数；为0时，选择寄存器数据；

（7）RegWrite：为1时写入寄存器堆，目的寄存器号是由RegDst选出的rt或rd，写入数据是由MemToReg选出的存储器数据或ALU的输出结果；

（8）ExtOp：符号扩展。为1时，符号扩展；为0时，0扩展；

（9）RegDst：目的地址，为1时，选择rd；为0时，选择rt。

**MIPS指令集**

本次实验共涉及三种类型的MIPS指令，分别为R型、I型和J型，三种类型的MIPS指令格式定义如下：

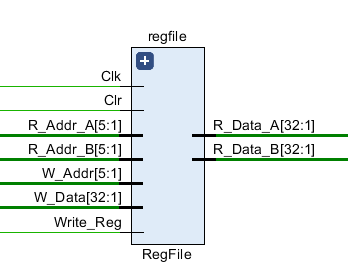
* R（register）类型的指令从寄存器堆中读取两个源操作数，计算结果写回寄存器堆；
* I（immediate）类型的指令使用一个 16位的立即数作为一个源操作数；
* J（jump）类型的指令使用一个 26位立即数作为跳转的目标地址（target address）；

1. 实验步骤

各模块设计

1. 寄存器模块

寄存器组是指令操作的主要对象，MIPS 处理器里一共有 32 个 32 位的寄存器，本实验中只声明一个包含 15 个 32 位的寄存器数组。读寄存器时需要 Rs，Rd 的地址，得到其数据。写寄存器 Rd 时需要所写地址，所写数据，同时需要写使能。以上所有操作需要在时钟和复位信号控制下操作。故寄存器组设计如下：

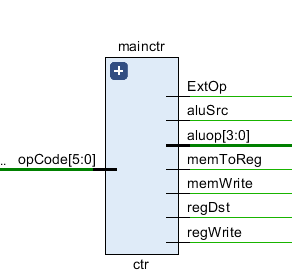
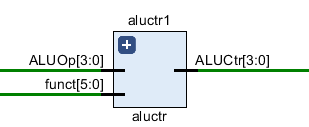


1. 控制器模块

根据指令中的指令码（op）和功能码（funct）的不同组合输出相应的控制信号

|  |  |  |
| --- | --- | --- |
| aluop[3:0] | 运算 | ALUCtr[4:0] |
| XXXX | J型指令 | x |
| 0000 | and | 00000 |
| 0000 | or | 00001 |
| 0010 | ori | 00001 |
| 0011 | Lw、sw、addi | 00010 |
| 0000 | add | 00010 |
| 0110 | bne | 00011 |
| 1000 | xori | 00100 |
| 0000 | nor | 00101 |
| 0000 | sub | 00110 |
| 0101 | beq | 00110 |
| 0000 | slt | 00111 |
| 0000 | sll | 01000 |
| 0000 | srl | 01001 |
| 0000 | sra | 01010 |
| 0000 | srlv | 01011 |
| 0111 | Lui | 01101 |
| 0000 | srav | 01110 |
| 0001 | slti | 10000 |
|  |  | 10001 |
| 0000 | xor | 10010 |

控制器设计如下：

1. ALU控制译码模块

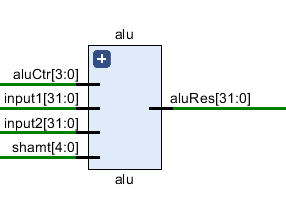
ALU 主要执行 5 种操作：与，或，加，减，小于设置。这五种操作可以使用四位的编码表示：0000，0001，0010，0110，0111。指令不同，则对应的 ALU 运算不同，所以该模块需要根据指令来控制ALU 进行正确的运算。

lw，sw，addi 指令均要求 ALU 执行加操作，则可分为一类，aluop编码 0011； beq 指令要求 ALU 执行减操作，则分为一类，编码 0101；

最后一类是 R 型指令，可以编码为 0000；但不同的R 型指令对应不同的 ALU 运算，故需要再通过指令的功能码进一步确定 ALU 的运算。

最终该模块即实现 4 位操作码以及 6 位功能码输出 4 位ALU 控制信号码。

1. ALU运算器模块



input1：操作数32位，输入；

input2：操作数，32位，输入；

ALUCtr：4位操作码，输入；

aluRes：运算结果，32位，输出；

zero：零标志，1位；当运算结果为0时，该位为1，否则为0；

1. 符号扩展模块

根据符号补充符号位

如果符号位为1，则补充16个1，即16'h ffff

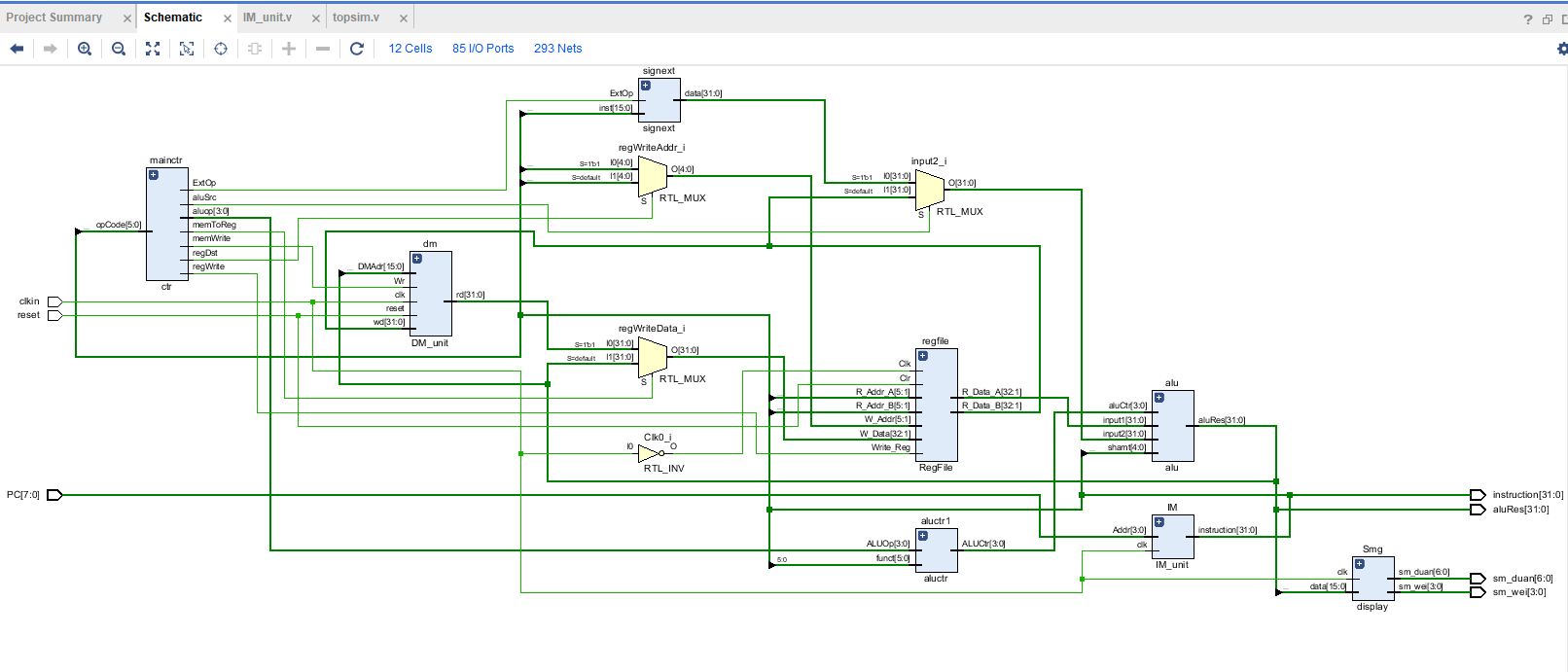
如果符号位为0，则补充16个0

1. 指令存储器模块

该模块有4位地址输入和32位数据输出，首先将16条指令写入存储单元中，然后根据4位地址输入选择相应的单元指令内容，将数据写入到输出变量中。

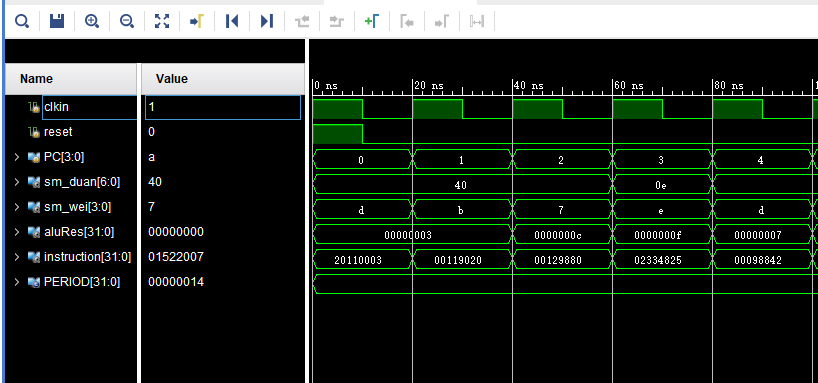
1. 数据存储器
2. 译码管显示模块
3. PC模块 top文件，组装各个模块

最终结构原理图



1. **实验结果**

**仿真图波形分析：**



指令1： 20110003 -> addi $s1, $0, 3

运算： s1 = 0 + 3 = 3

aluRes = 00000003

指令2：00119020 -> add $s2, $0, $s1

运算: s2 = 0 + 3 = 3

aluRes = 00000003

指令3: 00129880 -> sll $s3, $s2, 2

运算: s3 = s2 << 2

aluRes = 0000\_0000\_0000\_0000\_0000\_0000\_0000\_1100 = 0000000c

指令4: 02334825 -> or $t1, $s1, $s3

运算: t1 = 0000\_0000\_0000\_0000\_0000\_0000\_0000\_0011 |

0000\_0000\_0000\_0000\_0000\_0000\_0000\_1100

= 0000\_0000\_0000\_0000\_0000\_0000\_0000\_1111

=0000000f

aluRes = 0000000f

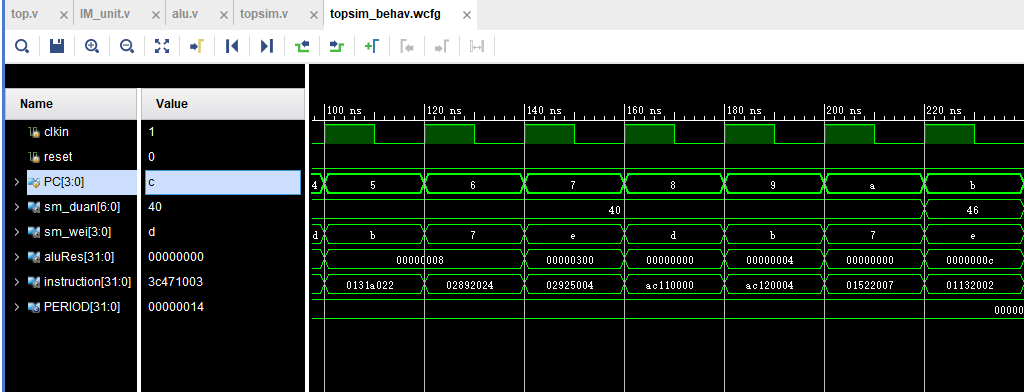
指令5: 00098842 -> srl $s1, $t1, 1

运算: s1 = t1 >> 1;

aluRes = 0000\_0000\_0000\_0000\_0000\_0000\_0000\_1111 >> 1

= 0000\_0000\_0000\_0000\_0000\_0000\_0000\_0111

=00000007;



指令6: 0131a022-> sub $s4, $t1, $s1

运算: s4 = t1 - s1 = 00000000f - 00000007 = 00000008

aluRes = 00000008

指令7: 02892024 -> and $4, $s4, $t1

运算: $4 = s4 & t1 = 00000008 & 0000000f

= 0000\_0000\_0000\_0000\_0000\_0000\_0000\_1000 &

= 0000\_0000\_0000\_0000\_0000\_0000\_0000\_1100

=0000\_0000\_0000\_0000\_0000\_0000\_0000\_1000

= 00000008

aluRes = 00000008

指令8: 02925004 -> sllv $t2, $s4, $s2

运算: t2 = s4 << s2

= 00000003 << 8

= 0000\_0000\_0000\_0000\_0000\_0000\_0000\_0110 <<8

= 0000\_0000\_0000\_0000\_0000\_0011\_0000\_0000

=00000300

aluRes = 00000300

指令9: ac110000 -> lw $0, 0($s1)

aluRes = 00000000

指令10: ac120004 -> sw $s2, 4($0)

aluRes = 00000004

指令11: 01522007 -> srav $4, $t2, $s2

运算: $4 = s2 >> t2

= 00000003 >> 00000300

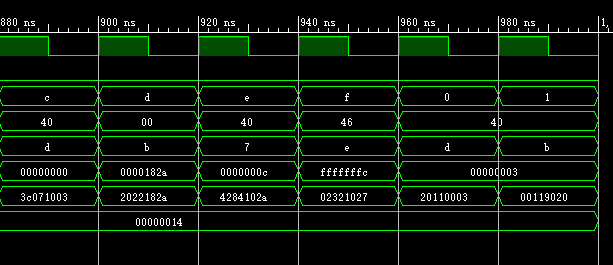
= 00000000

aluRes = 00000000

指令12: 11320002 -> beq $s3, $4, 2

运算: $s3 - $4 = 0000000c - 00000000 =0000000c

aluRes = 0000000c



指令13: 3c071003 -> lui

aluRes = 00000000

指令14: 2c22182a -> addi $1, $2, 182a

运算: $1 = $2 + 182a = 182a

aluRes = 0000182a

指令15: 4284102a -> slt $2, $s4, $4

运算 s4 - $4 = 0000000c -00000000 = 0000000c > 0

aluRes = 0000000c

指令16: 02321027 -> nor $2, $s1, $s2

! S2 = ~(00000007 | 00000003)

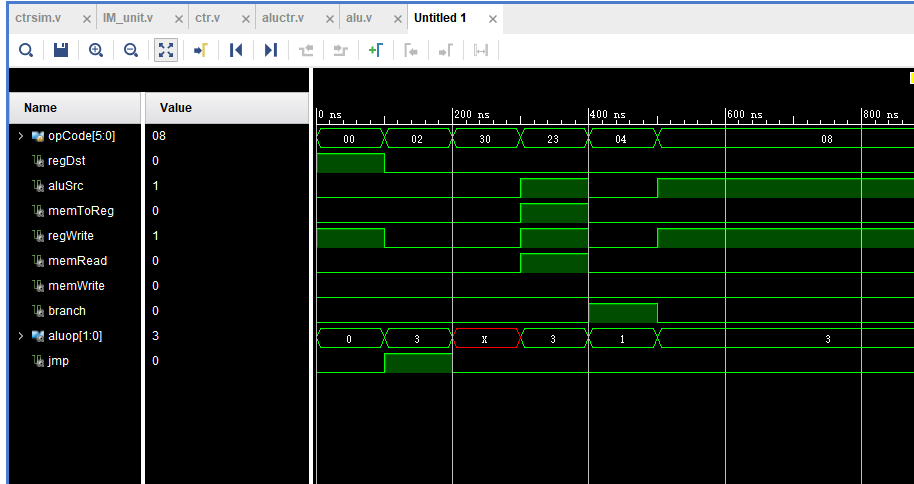
=~ (0000\_0000\_0000\_0000\_0000\_0000\_0000\_0011 |

0000\_0000\_0000\_0000\_0000\_0000\_0000\_0011)

= 1111\_1111\_1111\_1111\_1111\_1111\_1111\_1100 =fffffffc

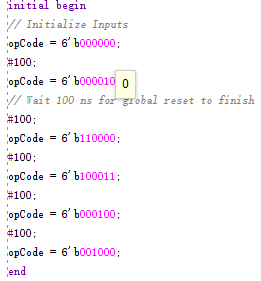
aluRes = fffffffc

控制器仿真

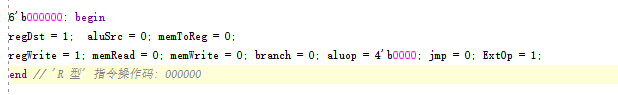


分析:

仿真部分代码



第一条 测试opCode = 000000, 对应ctr中的 R型指令设置, 相应结果正确



1. 测试 opCode = 110000, 因为 ctr中未定义此种op,故aluop显示X ,无对应项

其他测试,对应于ctr文件中,输出波形与结果均正确

**五、实验感想**

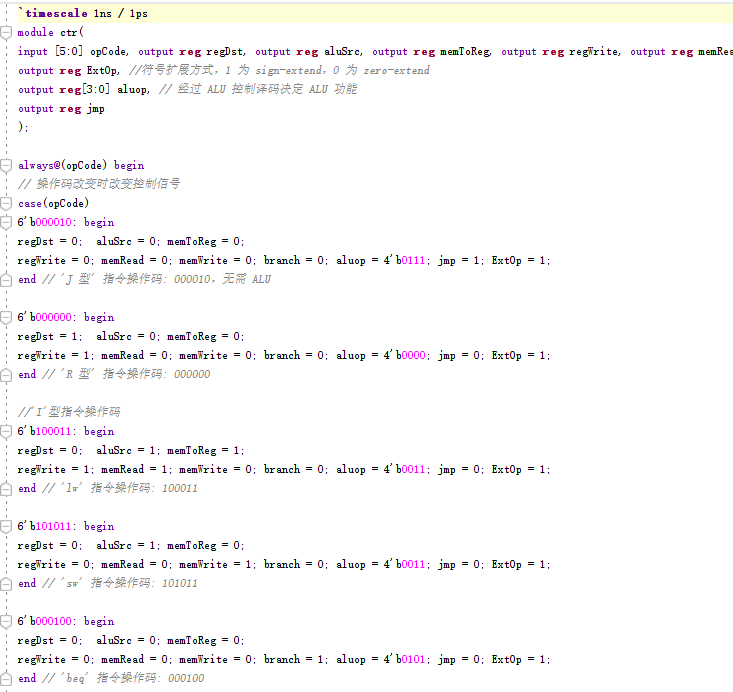
通过这次实验,我对CPU内部结构之间的关系更加明了,对指令的构成形式运算过程更加熟悉, 巩固了单周期CPU的相关知识, 发现了一些之前没有掌握的很好的知识,比如 aluop func 和aluctr之间的对应关系, 看书之后理清了, 收获还是很多的.

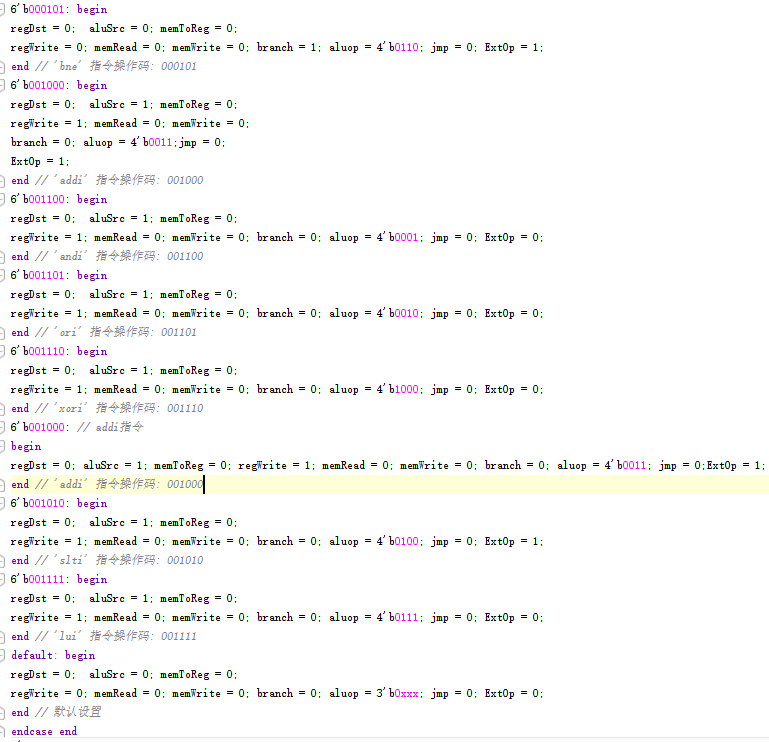
**附录（流程图，注释过的代码）：**

1. 寄存器模块

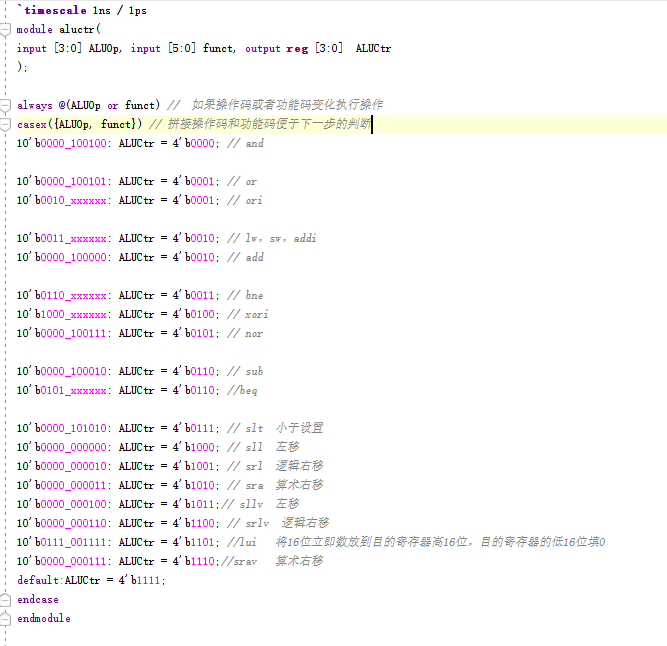


1. 控制器模块

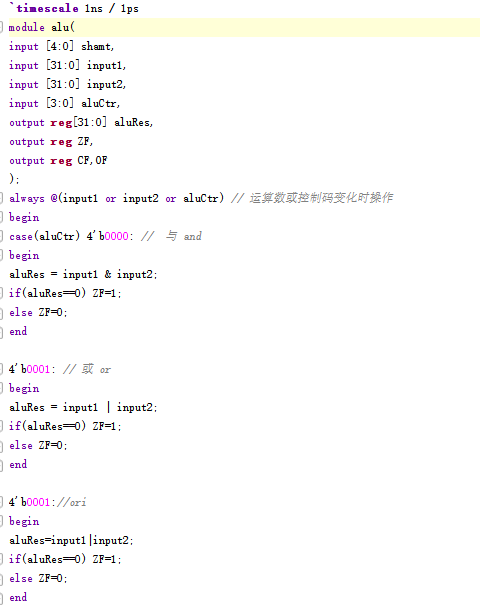


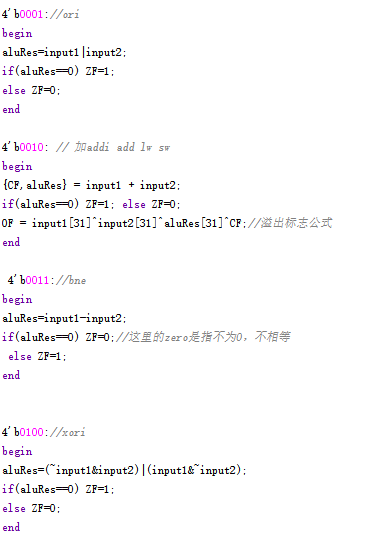


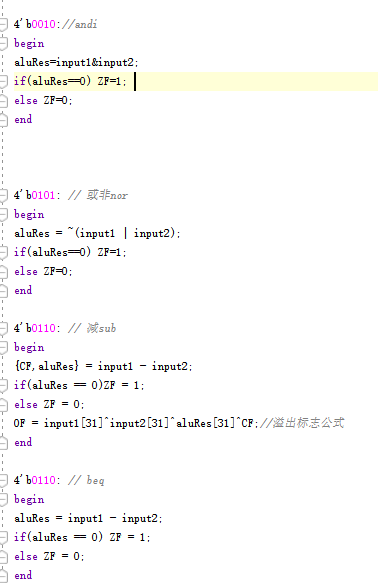
1. ALU控制译码模块

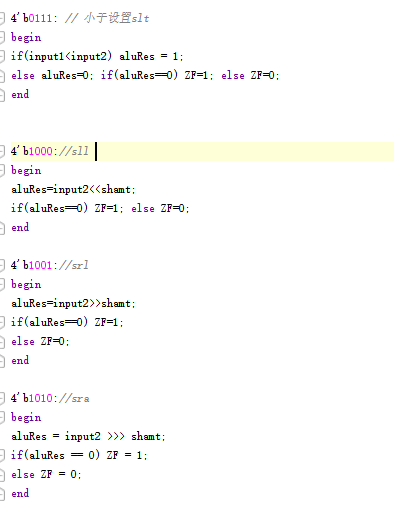


1. ALU运算器模块



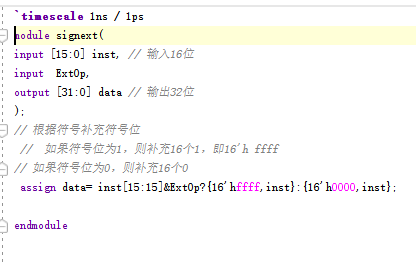




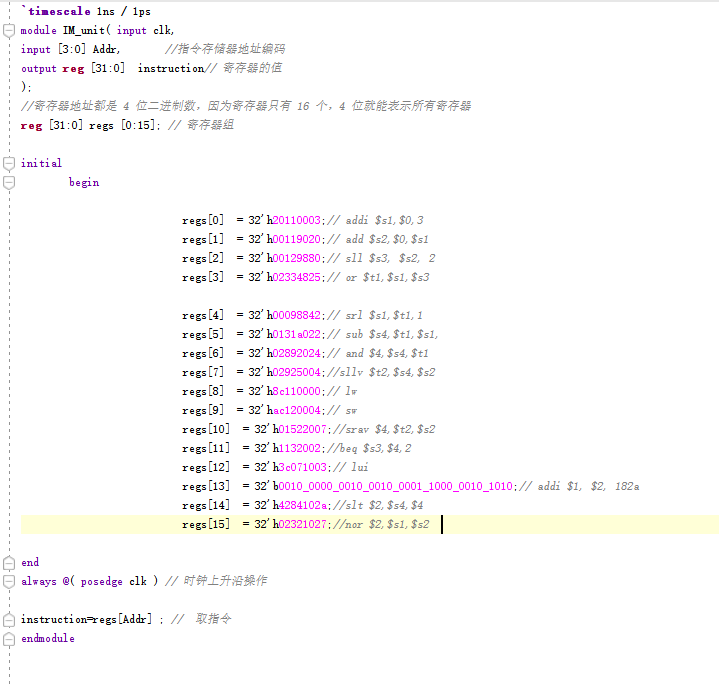




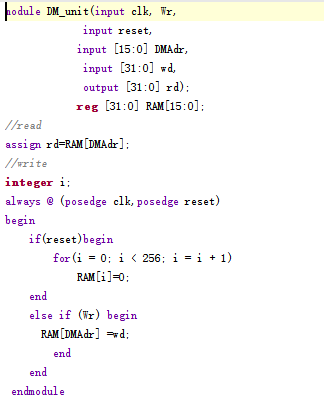
1. 符号扩展模块



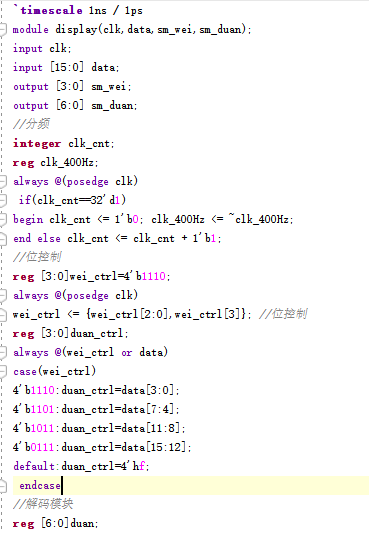
1. 指令存储器模块

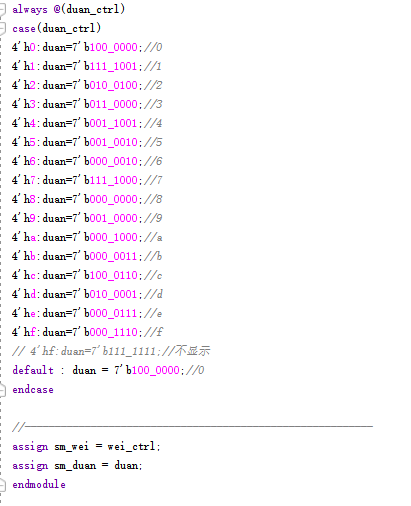


1. 数据存储器



1. 译码管显示模块





1. PC模块 top文件，组装各个模块



