**数字部件设计Project**

**单周期CPU电路设计解释文档**

1. **实验目的**

对RISC\_V架构单周期CPU的设计可以让学生深入理解RISC-V指令集的特点和指令格式,掌握RISC-V指令的基本操作和寻址方式。在对CPU基本电路的设计中，可以增强学生对CPU的工作原理,部件组成和连接的理解,掌握电路设计的基本方法和步骤。在实验的仿真测试中,学生能够提高电路设计和调试的实践能力,培养解决实际问题的能力。CPU的设计包含了很多时序逻辑与组合电路,在动手设计的过程中,学生可以进一步加深对时序电路与组合电路的特点,增强verilog代码的编写能力。

1. **实验内容**

1.实现指定RISC-V指令：

实验要求实现9个RISC-V指令：add、sub、or、slt、addi、ori、slti、sw、lw。每条指令长度为32bits。

2.设计CPU部件资源：

实验需要设计32个通用寄存器（X0-X31）、特殊寄存器PC（程序计数器）和指令暂存寄存器IR（指令寄存器），所有寄存器都是32bits字长。两个128bytes的存储器，采用little endian方式存储数据或指令。一个存放程序指令（最多32条指令），一个存放数据。

3.功能仿真与测试：

进行功能仿真,显示所有存储器件中的数据,通过功能仿真的结果来检验设计的正确性,并提交每条指令的执行仿真波形图及简要说明。

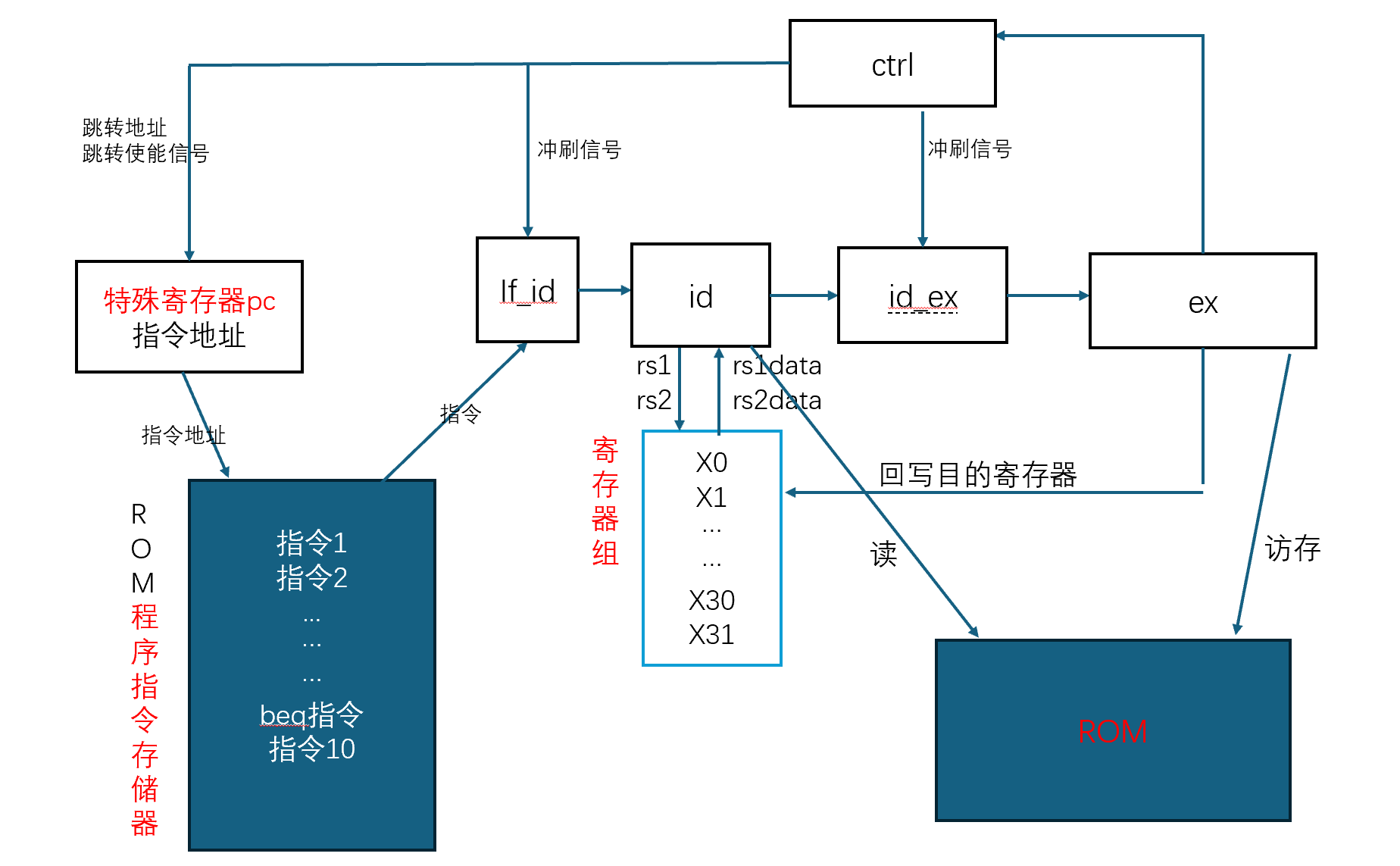
4.编写实验报告：

实验报告包括但不限于对功能仿真波形图的介绍和说明,指令处理的具体过程,以及对CPU所能完成的所有功能和相关性能分析。同时描述CPU的电路示意图、实现功能和性能改进和实验体会。

1. **设计思路和实现过程**
2. 总体设计思路

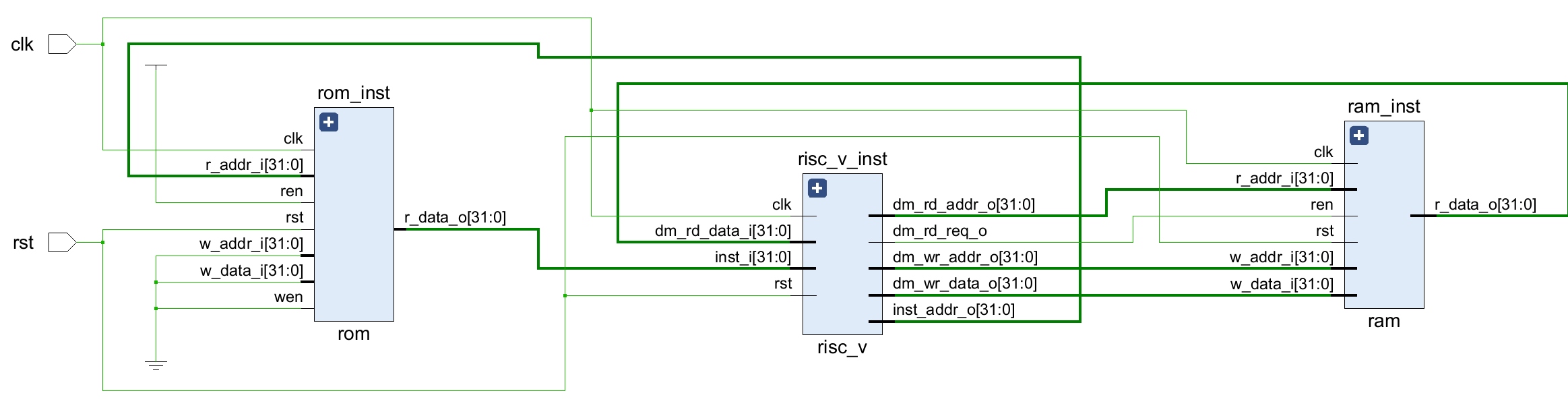
RISC\_V利于流水线设计,普遍为五级流水线设计,包括取值,译码,执行,访存,回写。在该project的实现中选择将执行,访存,回写放在一级,实现三级流水线设计。

1. CPU电路示意图
2. 各阶段模块连接示意

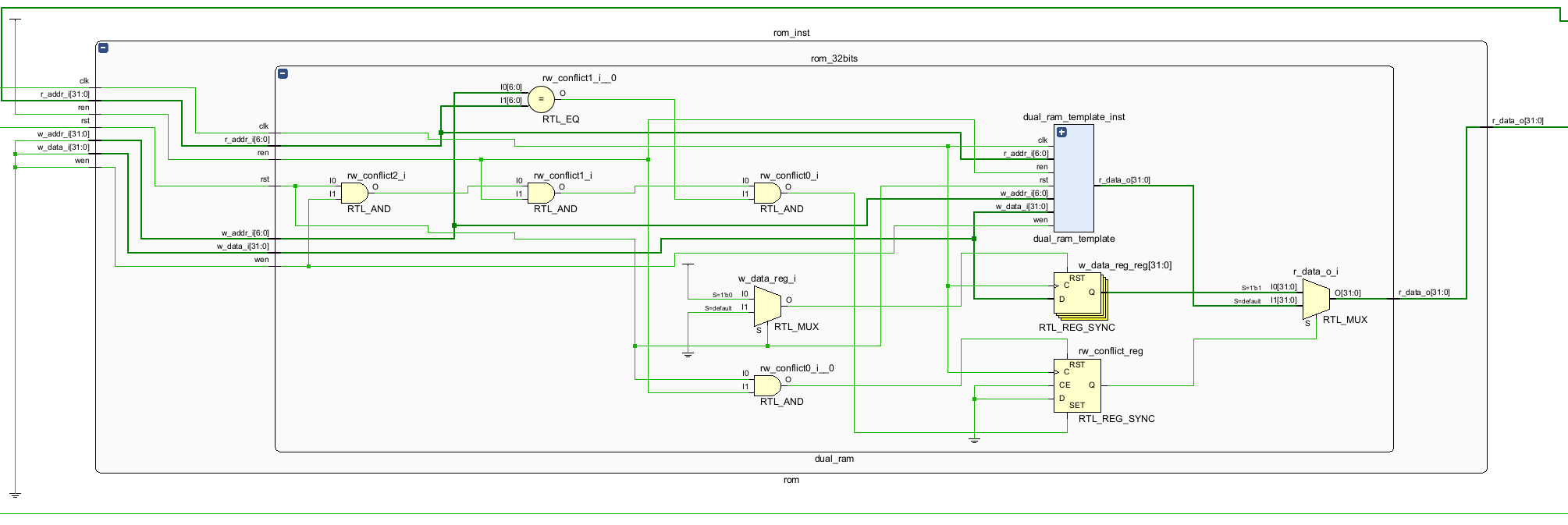


1. RTL Schematic

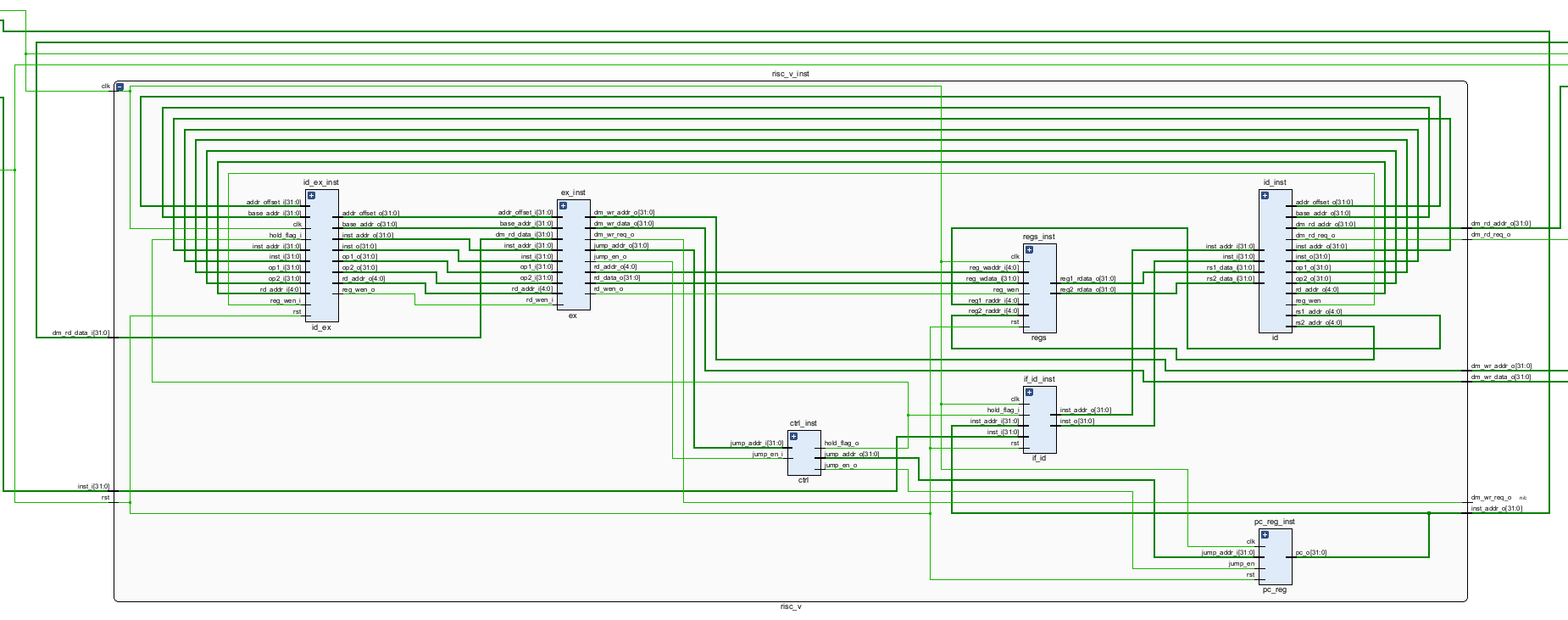
整体



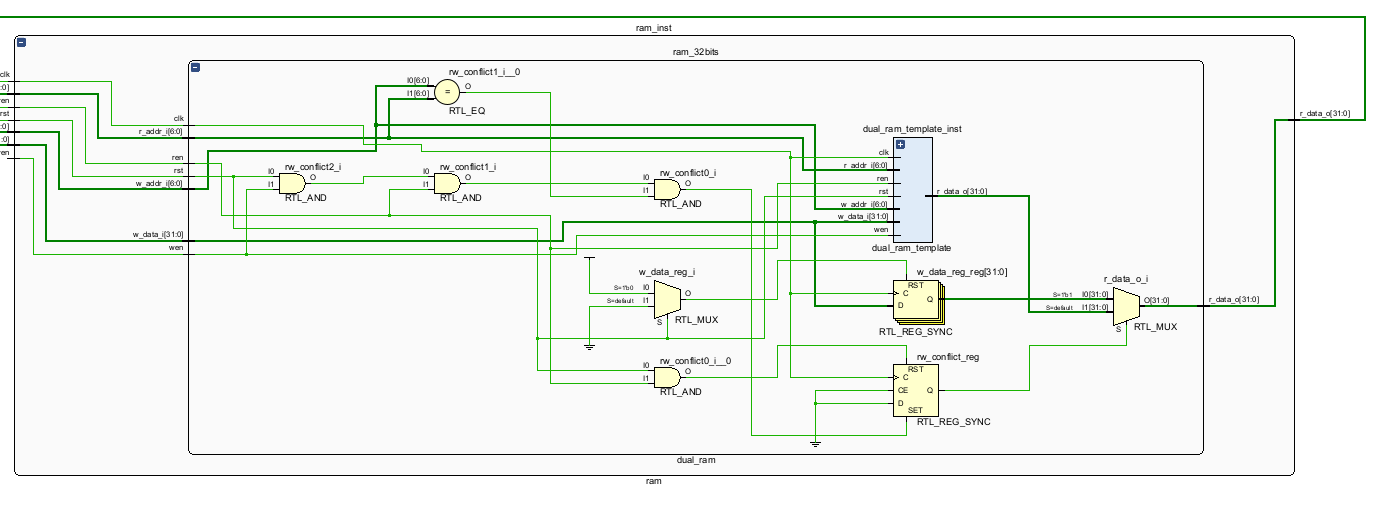
rom\_inst



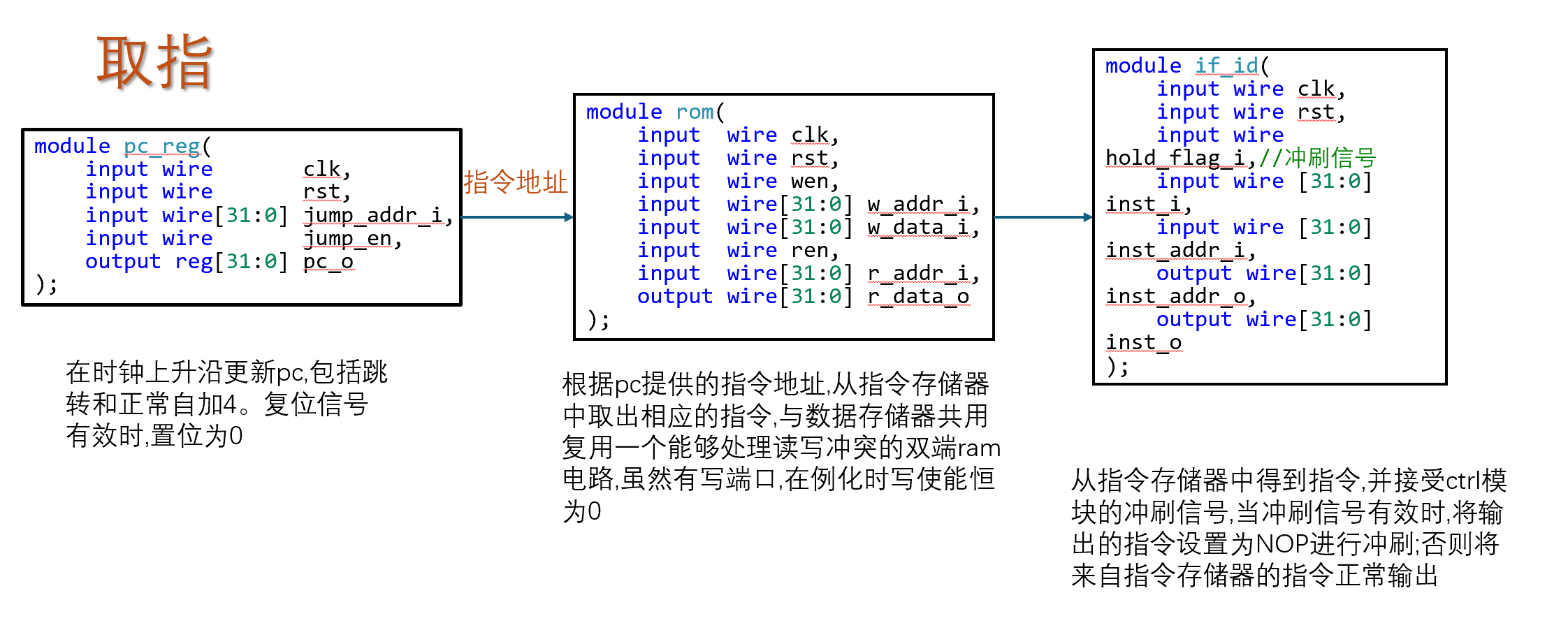
risc\_v\_inst



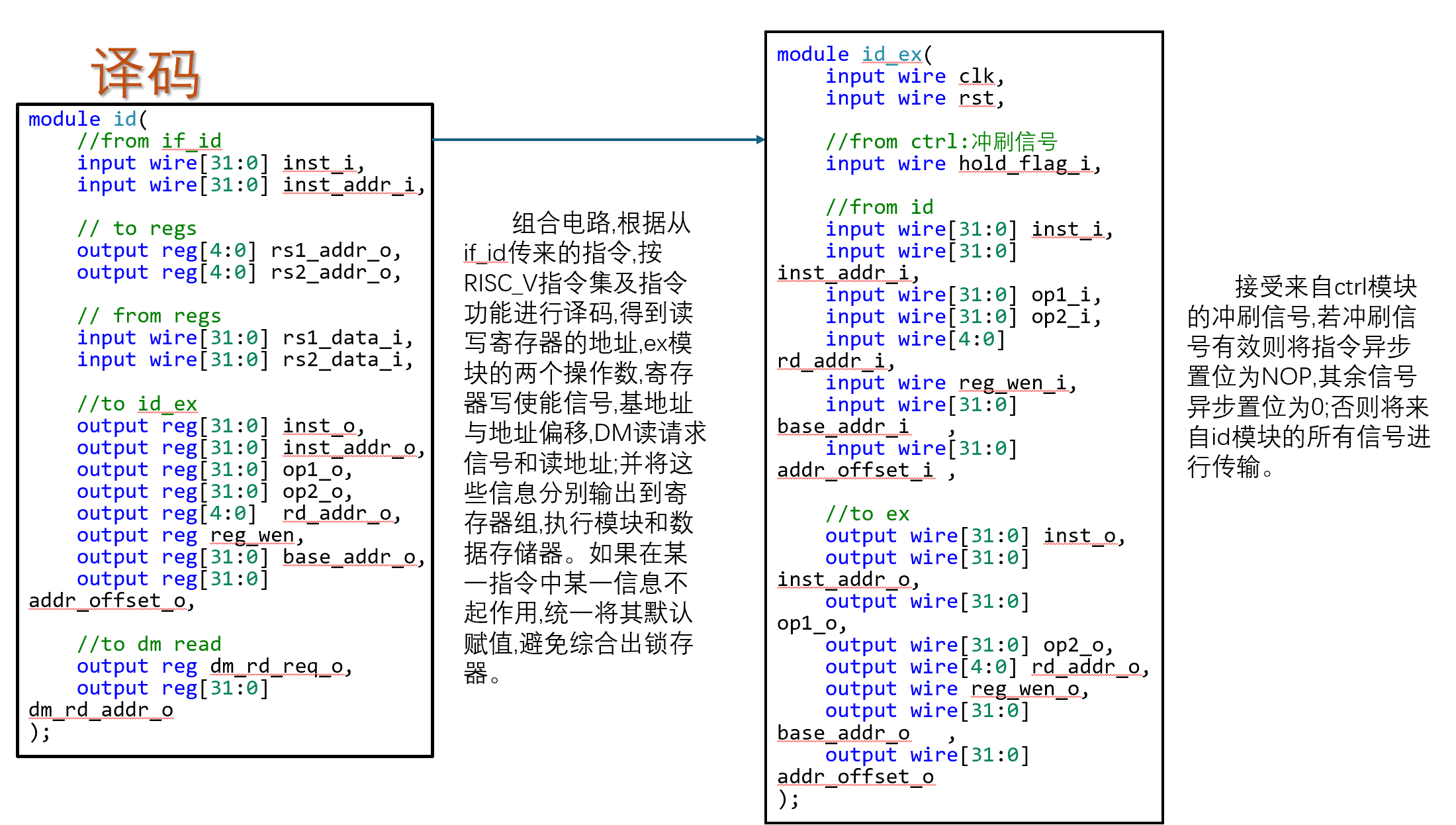
ram\_inst



1. 具体指令的设计执行过程
2. 取值



1. 译码



译码类型:先根据指令中的opcode大概分为5类

①I型指令:addi、ori、slti,根据指令中的func3对这三条指令进行区分,这类指令在译码时均需要得到读写寄存器的地址,执行(ALU)的两个操作数,寄存器组的写使能信号信息,其余信息 则进行默认赋值。

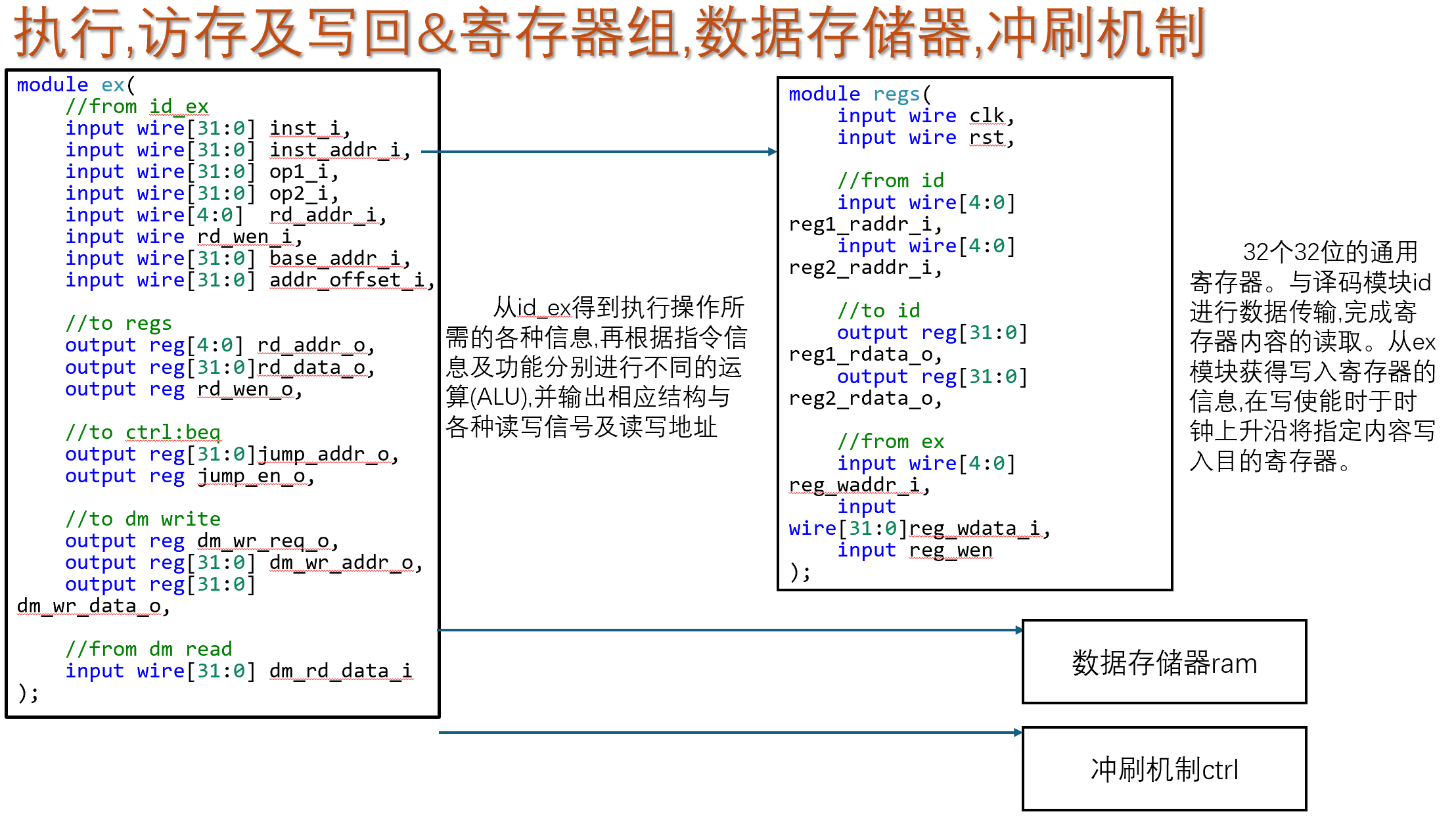
②R型指令:add、sub、or、slt,根据指令中的func3对它们进行区分,其中add和sub还需要通过func7进行区分。这类指令在译码时均需要得到读写寄存器的地址,执行(ALU)的两个操作数,寄存器组的写使能信号信息,其余信息则进行默认赋值。

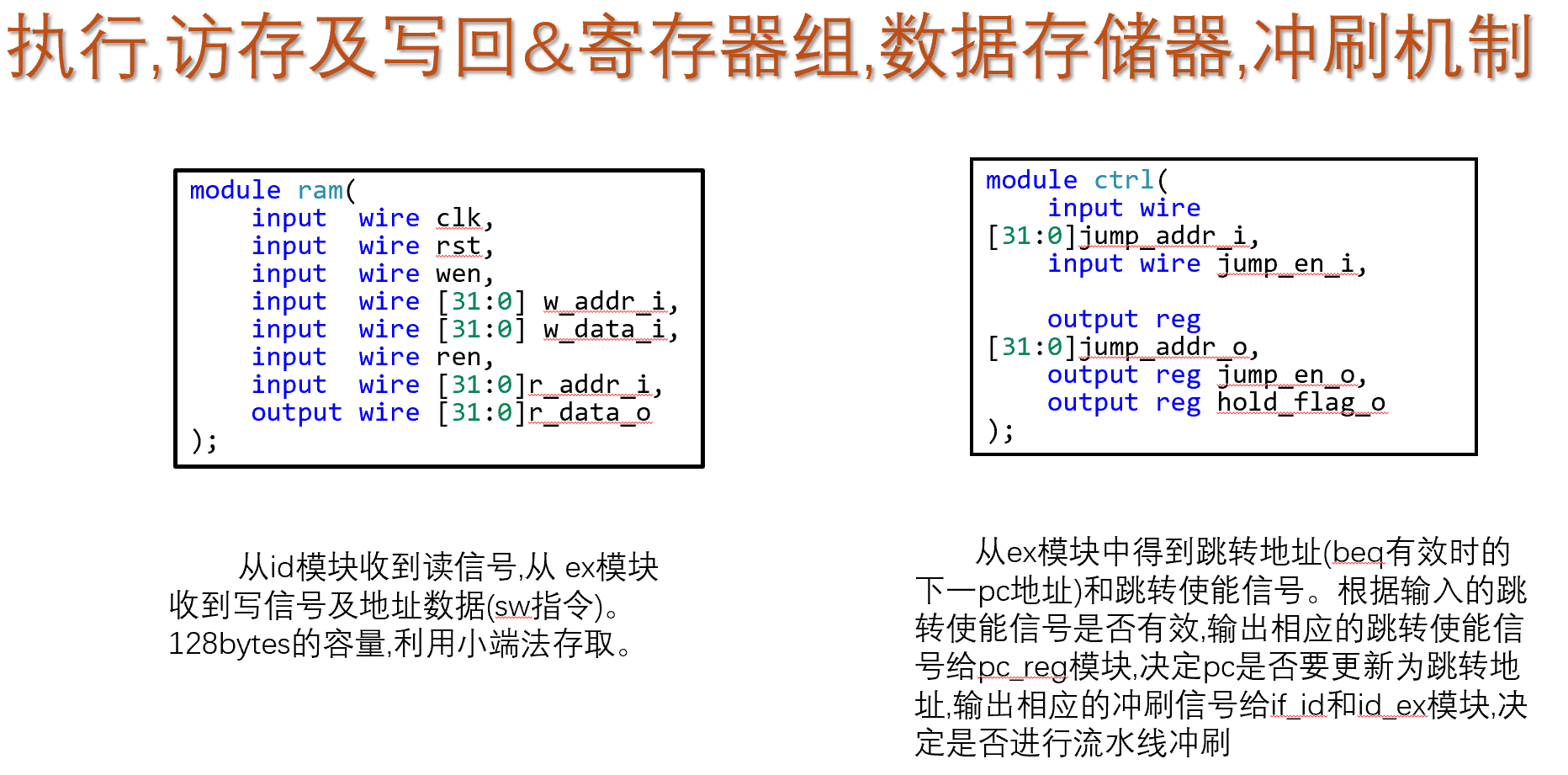
③B型指令(beq):译码时需要得到读取寄存器的地址,两个执行操作数(用于比较是否相等的两个寄存器数值),基地址(当前pc值)和偏移量(立即数)信息,其余信息则进行默认赋值。

④L型指令(lw):译码时需要得到数据存储器读请求信号及地址,源寄存器地址,写寄存器地址及使能,基地址(源寄存器数值)及偏移量(立即数),其余信息则进行默认赋值。

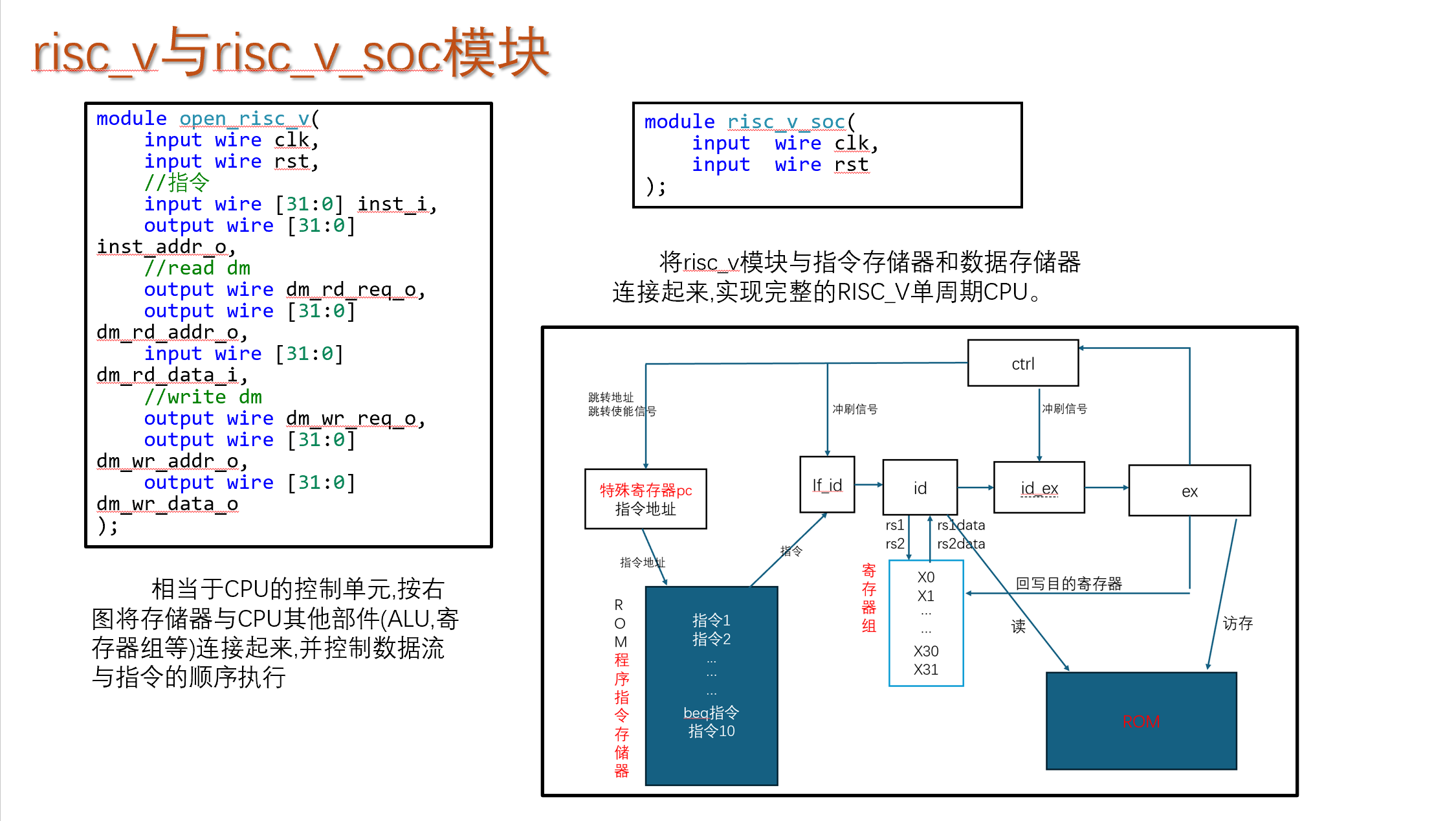
⑤S型指令(sw):译码时需要得到源寄存器地址,rs2内容作为操作数之一,基地址(源寄存器rs1数值)及偏移量(立即数),其余信息则进行默认赋值。

1. 执行,访存及写回





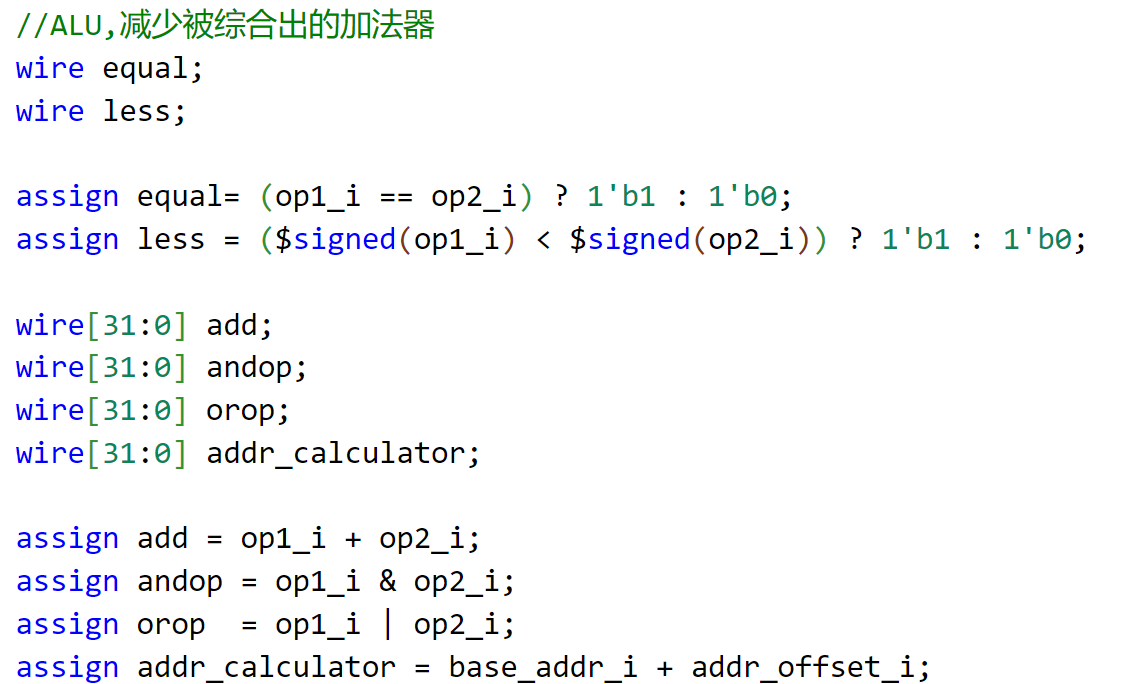
1. 部件连接

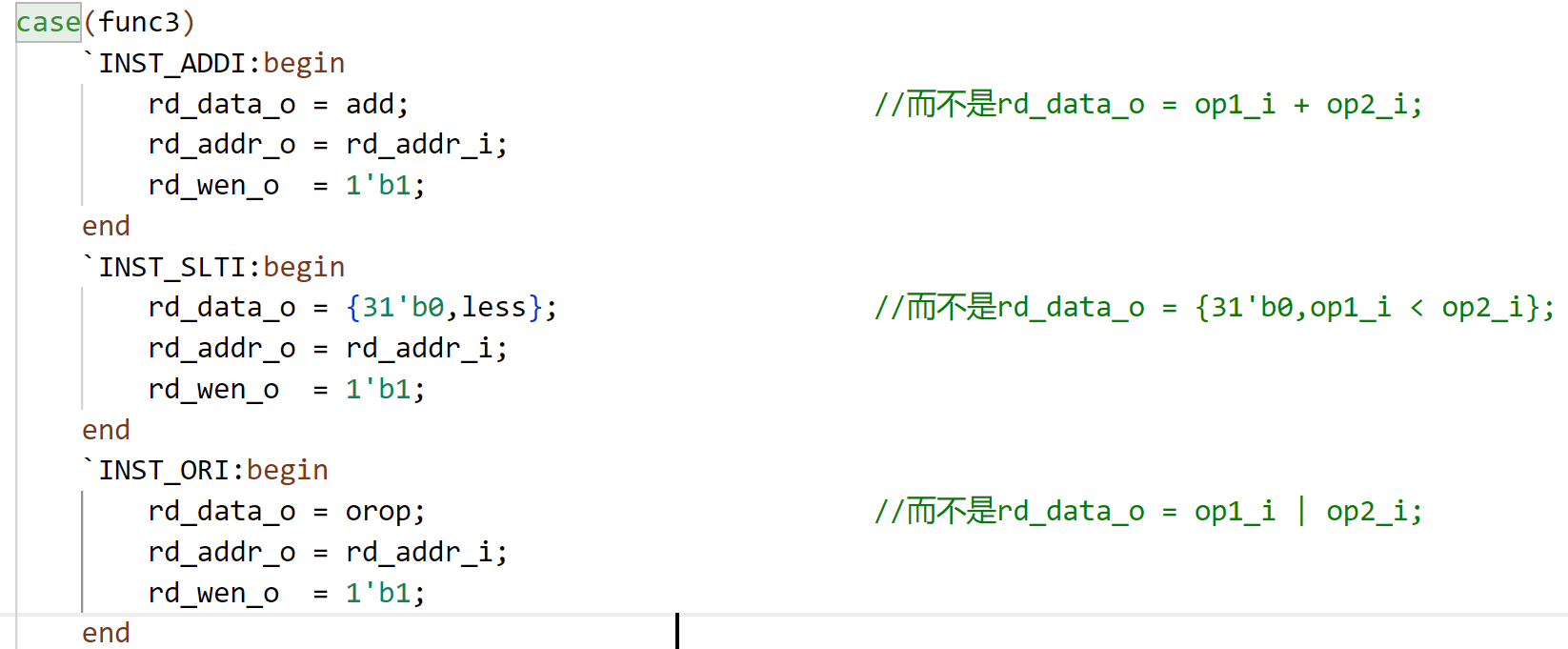


1. 性能改进及分析
2. 流水线设计

利用三级流水线设计提高了CPU吞吐量。

1. ALU设计

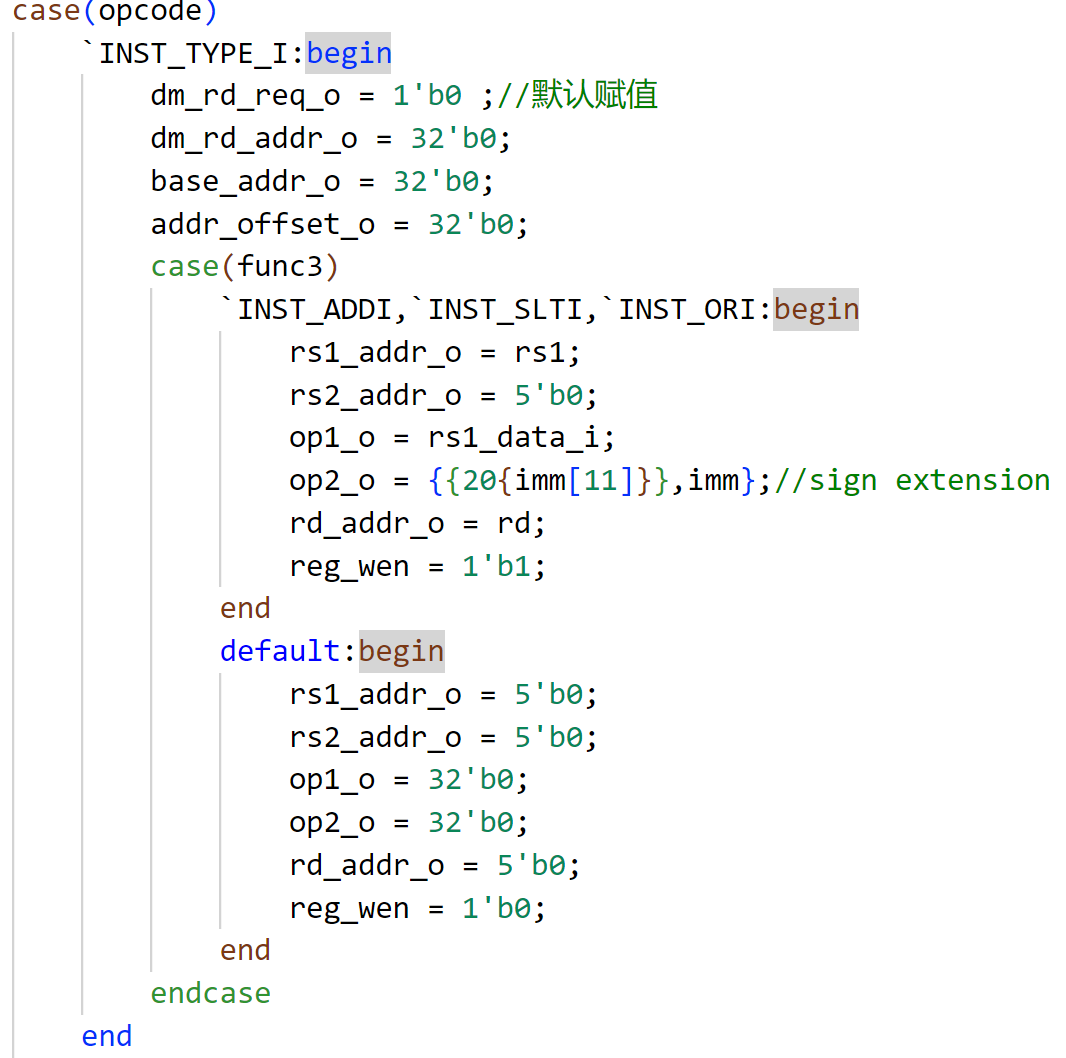




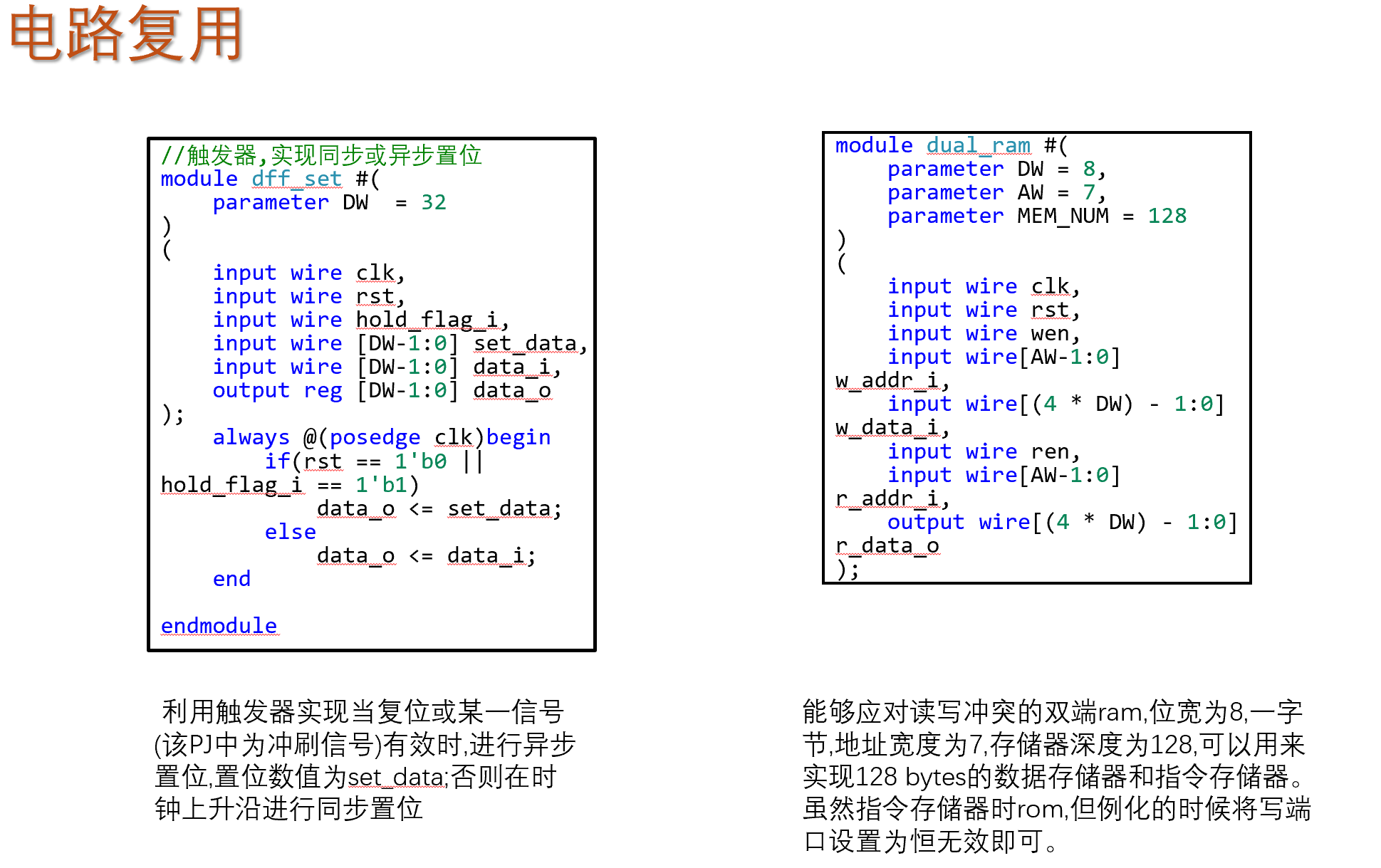
因为一个”+”就会被综合出一个加法器,因此采用ALU统一进行运算,可以降低综合的复杂度,从而提高性能。

1. 无锁存设计

在always @(\*)这种电平敏感的语句块中,如果所有的变量均能被赋值,将会综合出组合电路;否则综合出的电路将暗含锁存器。因此有必要对所有变量进行默认赋值,并写上default语句。

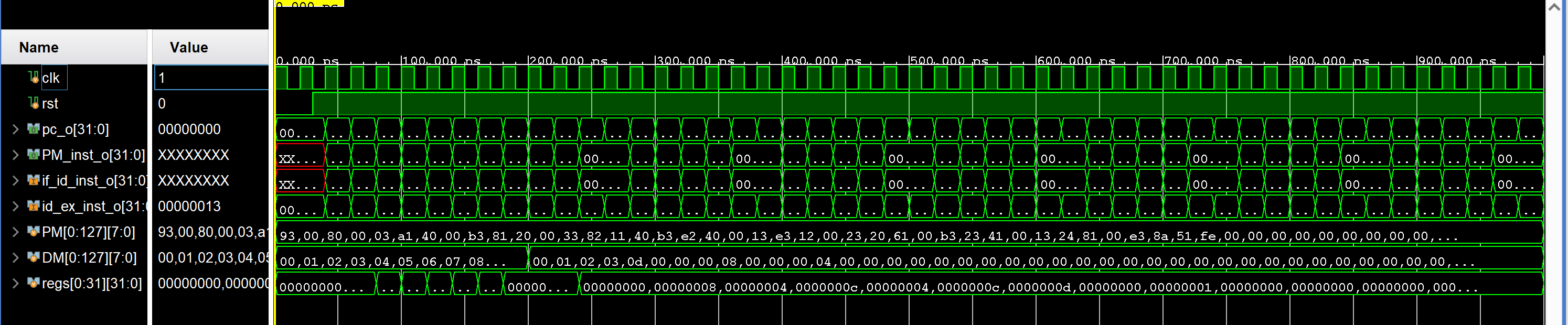


1. 电路复用



通过电路复用可以提高电路设计的可靠性与稳定性,从而降低设计复杂度与错误可能性。

1. **实验结果**
2. 总体指令执行仿真结果



1. addi X1, X0, 0x8 机器代码0x00800093

指令功能为将寄存器X0中的内容加立即数8后写到寄存器X1中。寄存器X0中内容恒为0,故写到X1的数据值为8。无分支跳转,该指令后的下一pc值为自加4。

X1 = 8 ,pc = 4

1. lw X2, 4(X1) 机器代码:0x0040a103

指令功能为读取DM中某一地址的32位数据,写到寄存器X2中,地址位X1中数据加偏移4,即0xc。无分支跳转,该指令后的下一pc值为自加4。

X2 = 4 , pc = 8

1. add X3,X1,X2 机器代码:0x002081b3

指令功能为将寄存器X1内容8与X2内容4相加后写到寄存器X3中。无分支跳转,该指令后的下一pc值为自加4。

X3 = 0xc , pc = 0xc

1. sub X4,X3,X1 机器代码:0x40118233

指令功能为将 寄存器X3内容12减去X1内容8的结果写入X4。无分支跳转,该指令后的下一pc值为自加4。

X4 = 4 , pc = 0x10

1. or X5,X1,X4 机器代码:0x0040e2b3

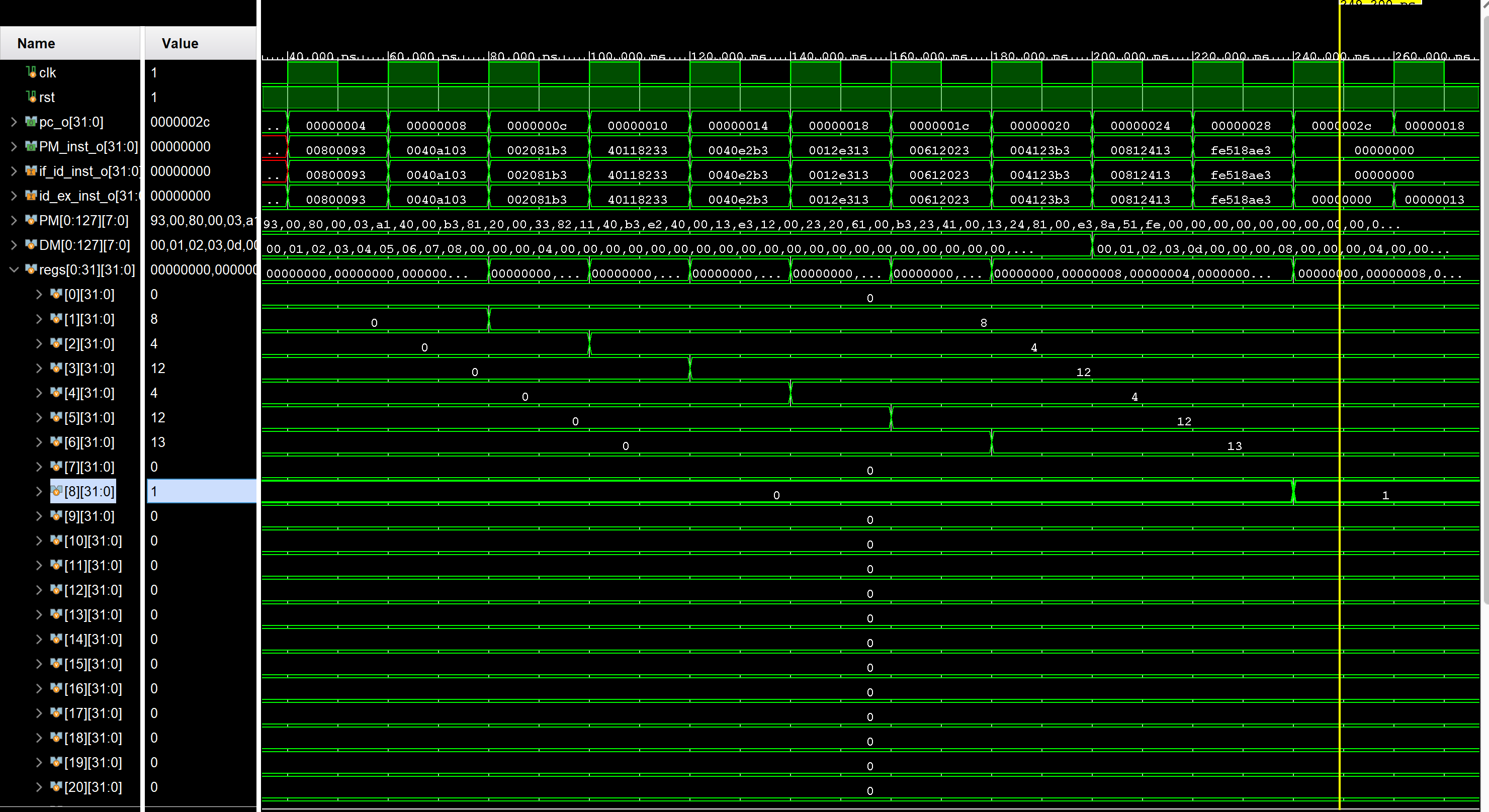
指令功能为将寄存器X1内容8与X4内容4按位或的结果写入X5中。无分支跳转,该指令后的下一pc值为自加4。

X5 = 0xc , pc = 0x14

1. ori X6,X5,1 机器代码:0x0012e313

指令功能为将寄存器X5内容0xc与立即数1按位或的结果写入X6中。无分支跳转,该指令后的下一pc值为自加4。

X6 = 0xd , pc = 0x18

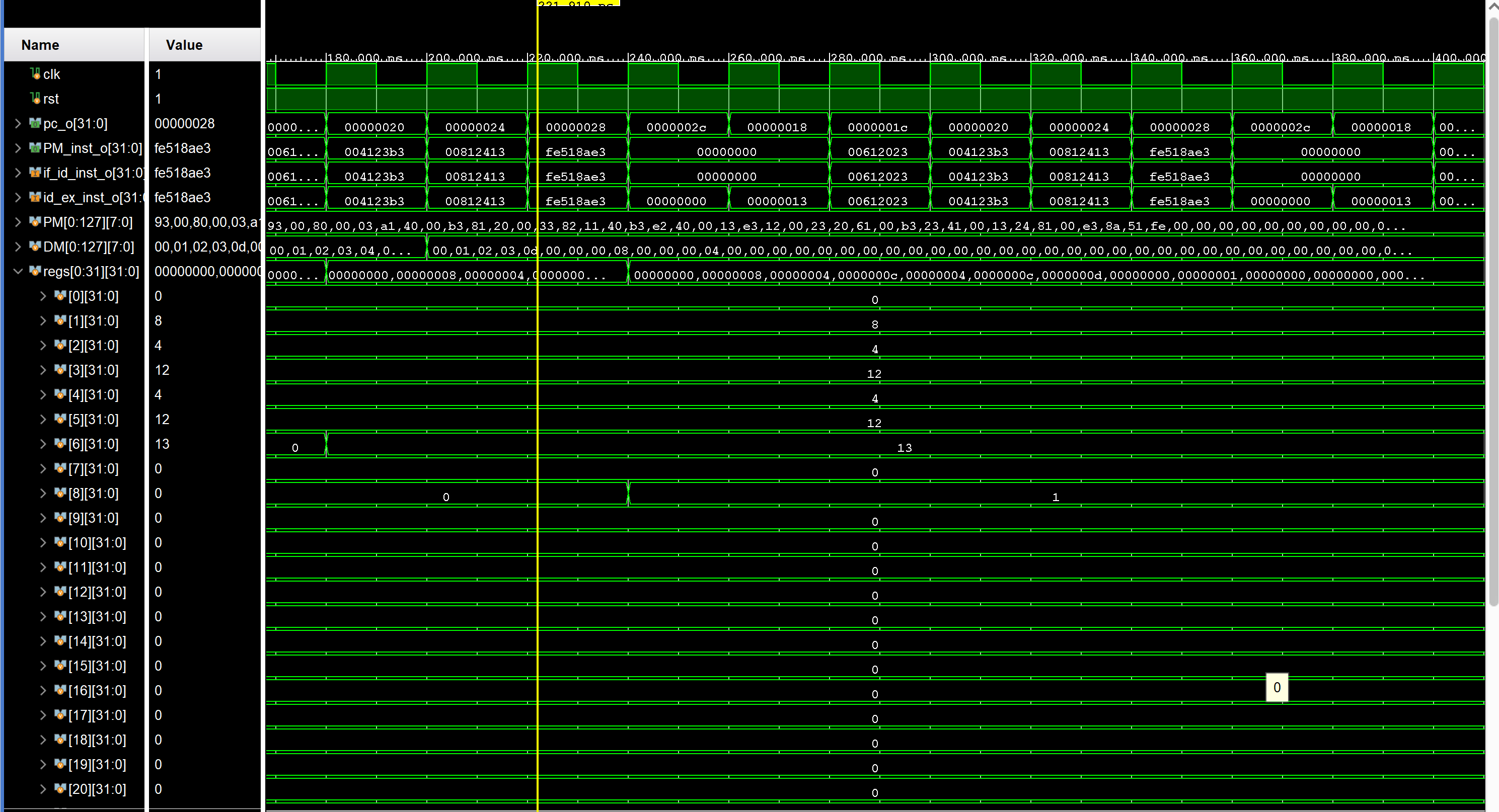


1. sw X6,0(X2) 机器代码:0x00612023

指令功能为将寄存器X6的32位内容0xd写入数据存储器DM中的某一地址。地址为X2中数据4加上偏移0。无分支跳转,该指令后的下一pc值为自加4。

Dm(0x4) = 0xd , pc = 0x1c





1. slt X7,X2,X4 机器代码:0x004123b3

指令功能为当寄存器X2内容4小于X4内容4时,将1写入X7;否则将0写入X7。无分支跳转,该指令后的下一pc值为自加4。

X7 = 0 , pc = 0x20

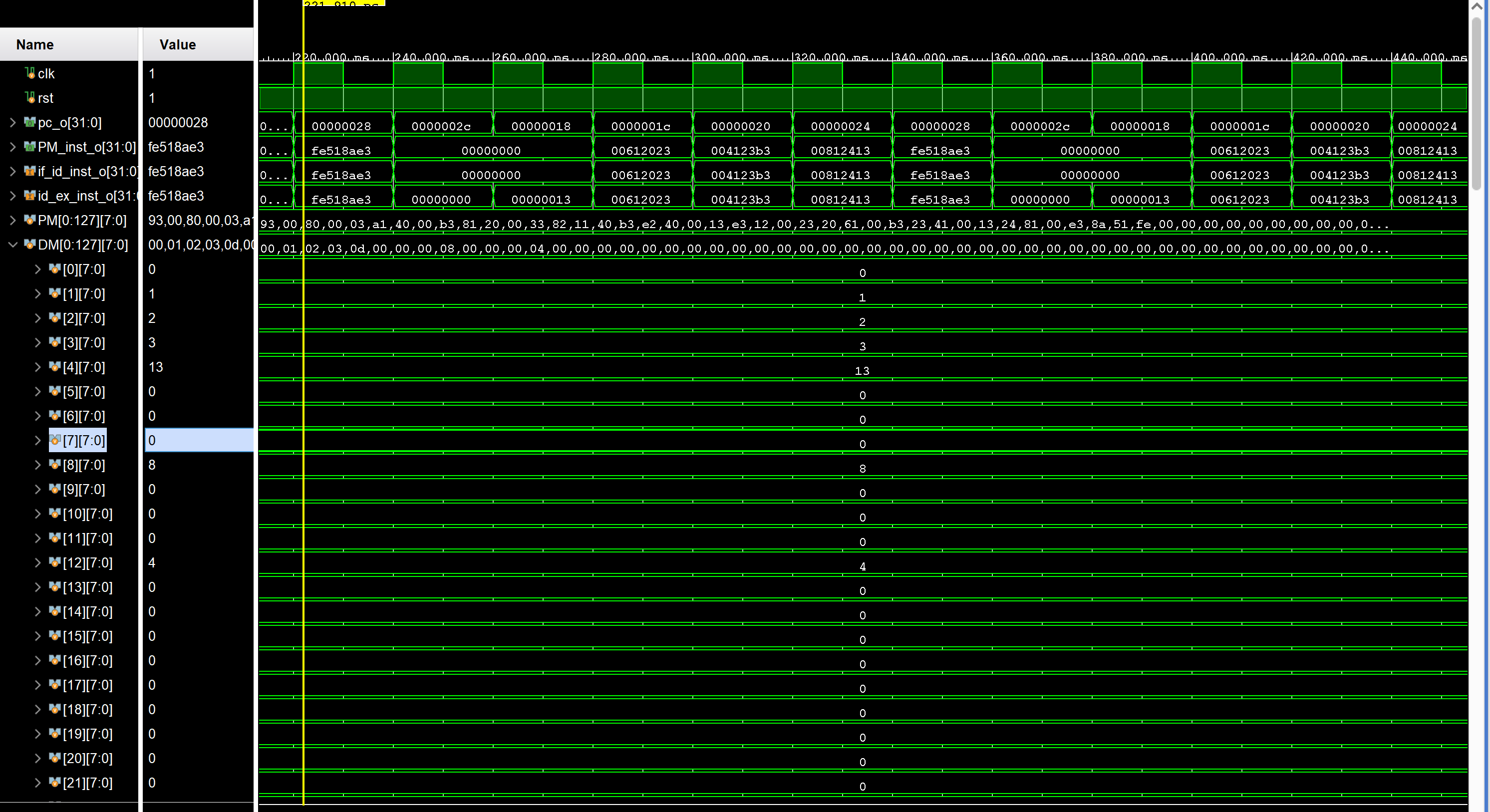
1. slti X8,X2,8 机器代码:0x00812413

指令功能为当寄存器X2内容4小于立即数8时,将1写入X8;否则将0写入X8。无分支跳转,该指令后的下一pc值为自加4。

X8 = 1 , pc = 0x24

1. beq X3,X5,-12 机器代码:0xfe518ae3

指令功能为当寄存器X3内容0xc与X5内容0xc相等时,分支跳转,下一pc值为自加立即数-12;否则下一pc值为自加4。产生分支跳转时执行地址0x18处的指令,无限循环地址0x18处到0x24处的指令直到仿真结束。



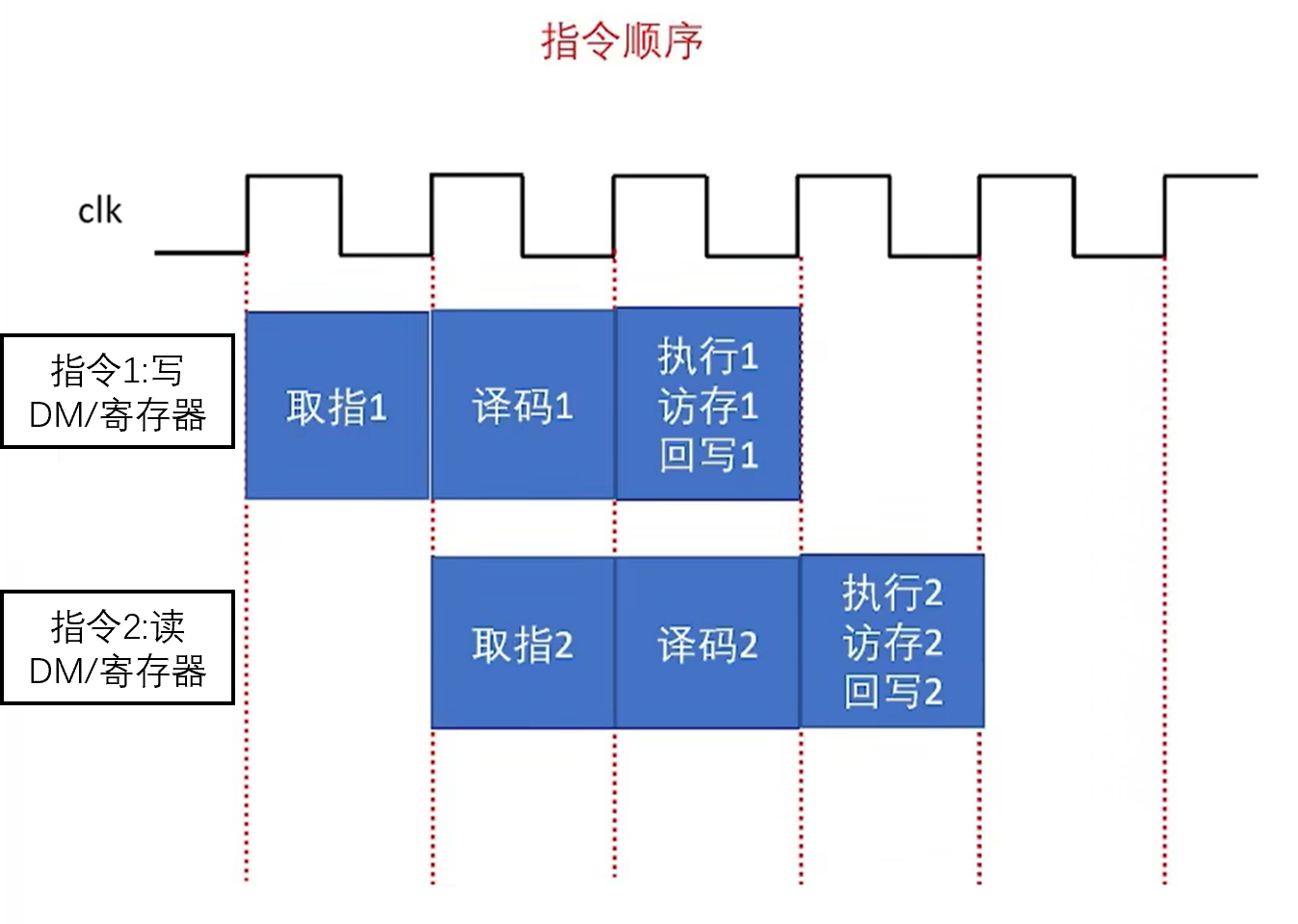
1. **遇到问题、解决方法和实验体会**
2. 异步与同步置位

当beq指令有效时,指令要进行跳转。但此时流水线上已有两条beq指令顺序地址之后的两条指令,应该怎么取消对这两条指令的执行?

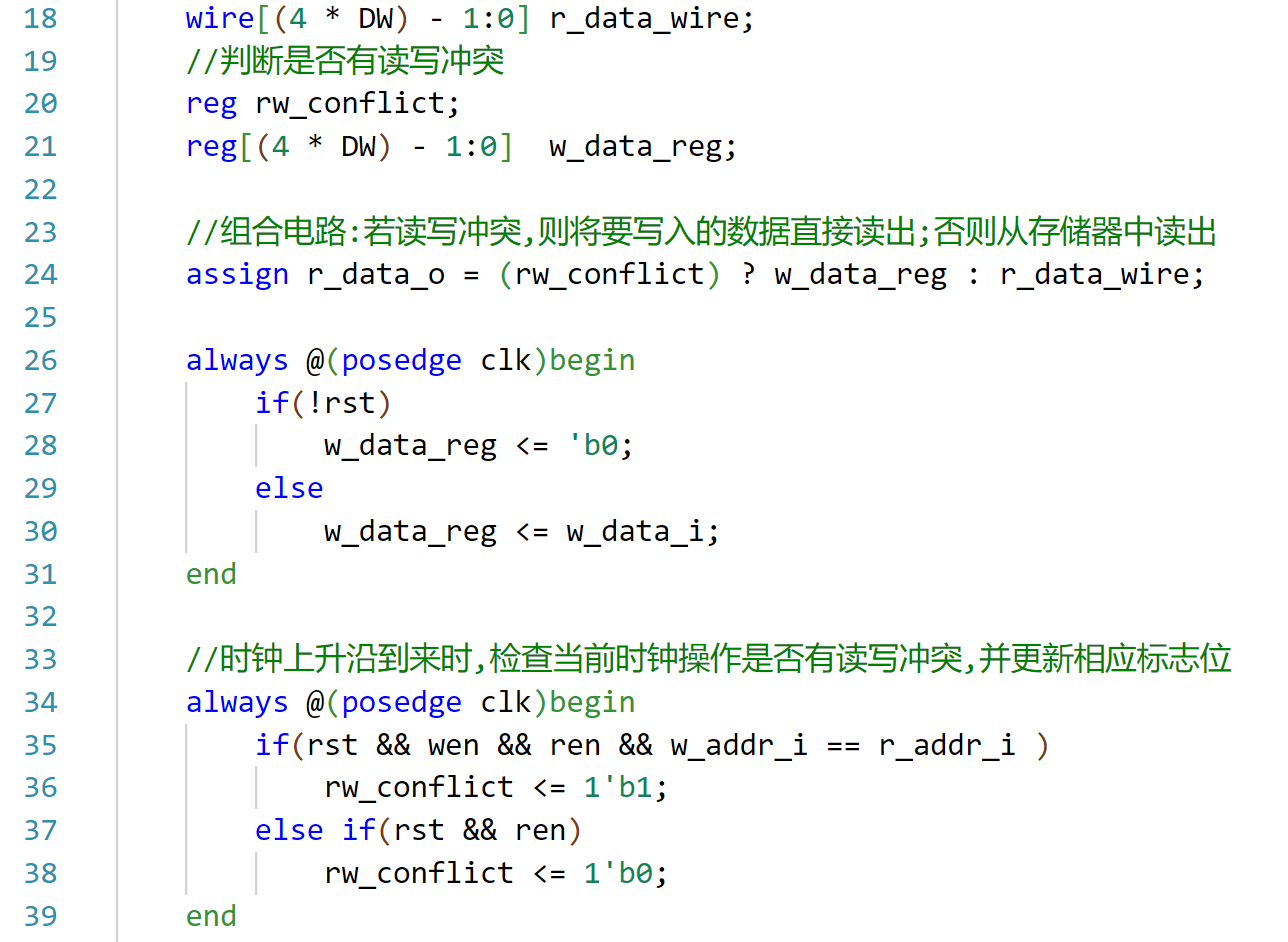
解决方法:利用冲刷信号,将这两条指令所处位置处的指令信号冲刷为NOP指令,即addi X0 X0 0。一旦冲刷信号有效就要对对应的指令输出重新赋值为NOP;无效则赋值为正常流水线上传输的指令性信号。故而可以使用触发器来实现这种异步与同步的置位。

1. 读写冲突

由于流水线的设计,在同一时钟周期可能同时发生对某一寄存器或数据存储器的写和读。如果上一条指令在该时钟周期进行执行,访存和写回阶段,而下一条指令在同样的时钟周期进行译码阶段,这样数据还未被写入就要被读出,此时将会读出旧值,发生读写冲突。读写冲突可能发生在寄存器组和数据存储器的读写中,要解决读写冲突,只需要在两者的写使能信号有效时,判断读出地址是否与写入地址相等,若相等,则发生读写冲突,直接把将要写入的数据读出;否则,正常从寄存器组或数据存储器中指定地址处读取数据即可。



双端ram中的读写冲突处理



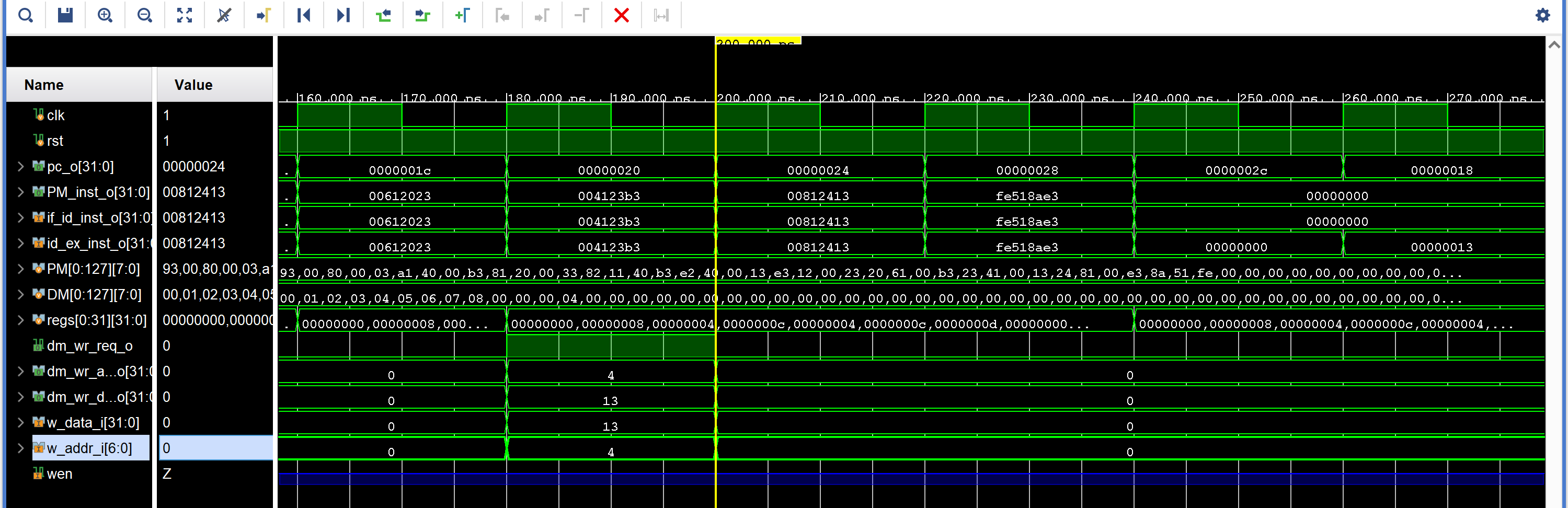
寄存器组中的读写冲突处理



1. 实例化连线

在仿真实验时,我遇到了寄存器组值全部符合实验结果,而DM中的值却未得到更新的情况,即指令sw X6,0(X2)没有被正确执行,该指令的执行效果为dm(0x4) = 0xd,但在最初的仿真结果中该地址的值并没有被正确更新,而是一直保持初始值。

我通过分析这条指令与上一条指令ori X6,X5,1的联系,发现执行这条指令的时候发生了读写冲突。我从regs模块开始逐步利用波形图查看其相应的输出信号是否正确。结果发现regs,id\_ex,ex模块中相应的输出信号均与预期中一致,说明问题很有可能与ram模块有关。在查看ram模块的相关信号时,发现其中的数据存储器写使能信号时钟为高阻抗状态。



我查看了ram模块的实现,并没有发现问题,我接着查看了risc\_v\_soc模块中关于ram实例化的连接,发现连接到数据存储器写使能信号的线网类型没有被声明。加上声明之后,写使能有效,数据存储器就呈现了预期的效果。

1. 实验体会

在动手设计一个简单RISC-V单周期CPU电路设计的过程中,我获得了丰富的实践经验和理论知识。实验开始前，我首先认真阅读了项目要求和参考资料，对RISC-V指令集的特点和单周期CPU的工作原理有了初步的认识。 之后,基于RISC\_V易于实现流水线设计的特点,我查阅了流水线设计的相关资料并结合之前计算机系统导论课程上关于处理器部分的相关知识,逐渐明晰了流水线阶段划分思路。在得到阶段划分与部件连接的大致框架后,我就开始逐步实现框架中的各个模块。

在设计过程中,我遇到了不少困难。例如如何解决读写冲突,怎样提高CPU性能,还有,在将各个模块进行组合时,我需要确保各个部件之间数据流动和控制信号的正确性。这要求我对整个电路的设计有清晰的认识。

为了克服这些困难,我不断查阅相关资料,参考经典的CPU设计,并进行了多次仿真和调试。在这个过程中,我逐渐掌握了数字电路设计和仿真的方法,提高了自己通过调试找出代码问题的能力。我也深刻体会到了理论与实践相结合的重要性。只有将理论知识应用到实践中，才能真正理解和掌握它。

1. **文件清单**
2. Readme.docx : 每条指令的执行仿真波形图（加简要说明),实验报告和文件清单
3. 文件夹RISC\_V:所设计CPU的Vivado project文件
4. 文件夹code:项目中所用到的所有v文件(包括激励文件)和存储器初始化数据文本文件



各文件从上到下依次为:

流水线冲刷机制实现模块

宏定义(指令集)

复用电路 异步与同步置位

数据存储器初始化数据

复用电路 能处理读写冲突的双端ram

执行模块

译码模块

译码\_执行模块

取值\_译码模块

PC寄存器

指令存储器初始化数据

DM模块

寄存器组模块

CPU控制单元

CPU

PM模块

激励文件