# 计算机体系结构期末综合实验报告

| 实验名称 | 五级流水线优化工作 | 班级 | 李雨森老师 | 指导老师 | 董前琨 |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 | 辛杰 | 学号 | 2213034 |  |  |
| 实验地点 | A306 | 实验时间 | 2024/11/20 |  |  |

[计算机体系结构期末综合实验报告](#计算机体系结构期末综合实验报告)  
 [一、实验目的](#一实验目的)  
 [二、实验任务](#二实验任务)  
 [三、前期准备](#三前期准备)  
 [1、将同步指令ROM和数据RAM替换为异步](#Xe91d0ec5617e95fb7fd643a4dec5f3fd1a2e218)  
 [四、实验内容](#四实验内容)  
 [1、旁路技术](#X58722aca8737cb2b6aabecd050e7a3e68abf253)  
 [2、指令拓展](#X78d005fbc981cacd627c69c424cb1490dfbb13d)  
 [1）R型指令](#X0e834a45cfd4f38dd8b70a0546622b62870a0b5)  
 [无符号除法器设计](#无符号除法器设计)  
 [核心逻辑：逐位除法](#核心逻辑逐位除法)  
 [无符号除法指令实现](#无符号除法指令实现)  
 [无符号除法指令格式](#无符号除法指令格式)  
 [1. 扩展 ID 到 EXE 的总线](#X23c21f7d39fec4e7fb33e3305ceba4bd3503599)  
 [2. 定义控制信号](#X524b73505c8cf10c0c925935fdd327076d9def0)  
 [3. EXE 阶段处理逻辑](#X8d95fb0fb5da54f5db958d3ad722353844b52ad)  
 [4. 指令格式与执行](#X41d892a8fc0ac0632078b0e95acf11f19cbb00c)  
 [无符号取模指令实现](#无符号取模指令实现)  
 [无符号取模指令格式](#无符号取模指令格式)  
 [1. 扩展总线](#X20bdbbcb482270cb85bc4078a151c3cd06ab78e)  
 [2. 定义取模控制信号](#X3c178b5ba5d501587334282c1387c853e655c71)  
 [3. EXE 阶段处理逻辑](#X08a36789bc0061ea6bf139280fae6f832363afb)  
 [4. 指令格式与执行](#X4549d3d1210c2958aef08b0c4f088239f518b66)  
 [2）J型指令](#X81c68dfe5fdbfc6a800f9158a6dc0ab0a209fc2)  
 [JALX 指令格式](#jalx-指令格式)  
 [实现步骤](#实现步骤)  
 [修改 decode.v 文件](#修改-decodev-文件)  
 [操作数设置](#操作数设置)  
 [更新指令内存](#更新指令内存)  
 [3）I型指令](#X4e91e433c646817044cd0029ecbc40e0d32b67f)  
 [无符号立即数除法指令格式](#无符号立即数除法指令格式)  
 [无符号立即数取模指令格式](#无符号立即数取模指令格式)  
 [解码逻辑修改](#解码逻辑修改)  
 [1. 定义控制信号](#Xbe664df61d71d0e5251766ba411bcbc047415b4)  
 [2. 更新立即数扩展逻辑](#Xaf8d996273755376a9a4b525860bfb4cd7a3cbe)  
 [3. 更新目标寄存器选择逻辑](#Xdfd802721a3b317e8372dbc515eccbe79c50482)  
 [4. 确认指令是否需要 rt](#X707746e96774d5943a323f780665d5a5db62d6a)  
 [5. 合并除法和取模控制信号](#X27ecf01a7635ede2740b0c54ad3d454c9a4d704)  
 [五、程序展示](#五程序展示)  
 [1、旁路技术](#X3049e39cd910c462e528782d01558b7b9f68c74)  
 [2、指令拓展](#X35d1c840b8b6ee0ec14d793d9d4ae27e52492af)  
 [1）R型指令](#Xcc6a4eb0caaa2dacdb9f691144aa6d9066839be)  
 [无符号除法](#无符号除法)  
 [无符号取模](#无符号取模)  
 [2）J型指令](#X2919c1f6b50e1a716fcd8afb81d480d05e02c46)  
 [3）I型指令](#X9b1c20bd4478ad72f1707b24cb3363d4c1c89de)  
 [无符号立即数除法](#无符号立即数除法)  
 [无符号立即数取模](#无符号立即数取模)  
 [六、思想感悟](#六思想感悟)

## 一、实验目的

1. 加深对计算机组成原理和体系结构理论知识的理解。
2. 培养对 CPU 设计的兴趣，在理解现有 CPU 架构的基础上，引发对体系结构的思考和创新。
3. 培养创新思维能力，并通过实践验证新想法。

## 二、实验任务

1、完成旁路技术和前向传递技术的改进（40%）

2、完成对目前流水线CPU的指令扩展，要求I型，J型，R型至少各扩展1条指令。（20%）

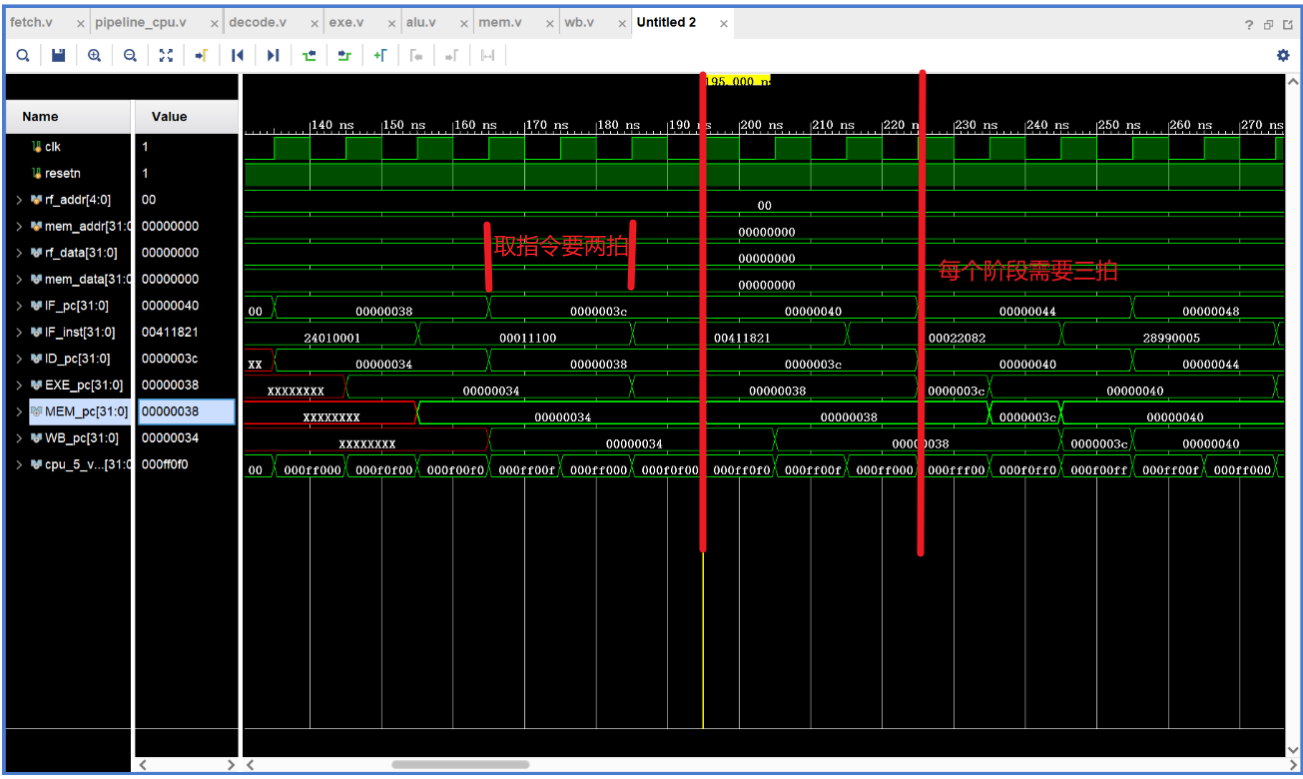
3、按照《CPU设计实战》这本书中的内容，依次完成异常和中断、AXI总线、TLB和Cache功能的添加。（60%）

**说明：本次实验完成了1和2**

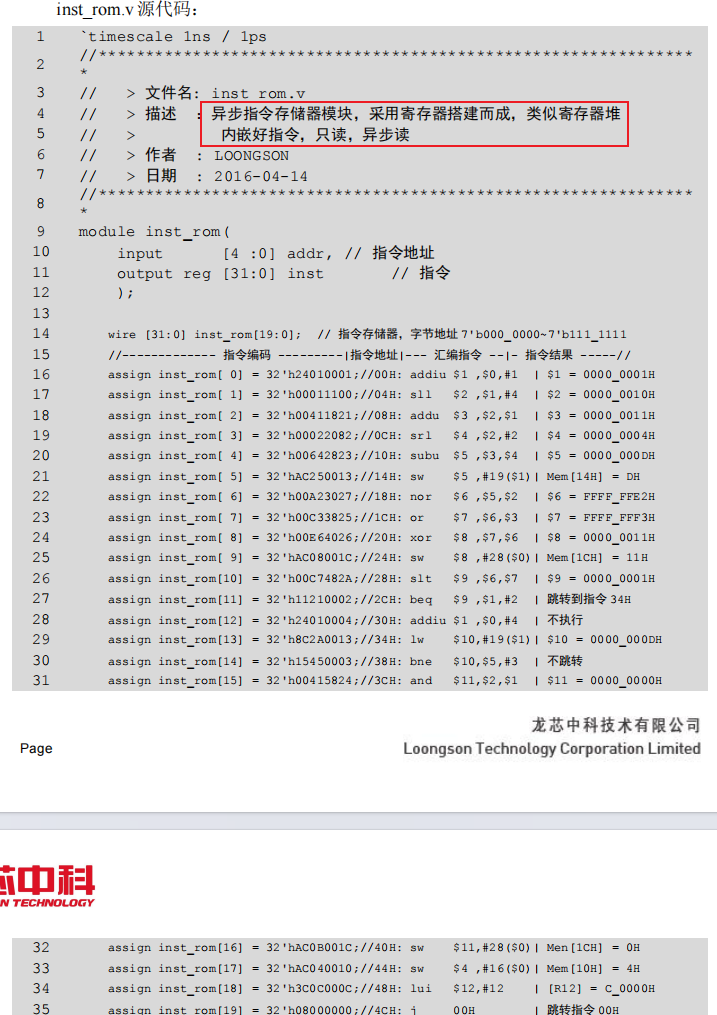
## 三、前期准备

### 1、将同步指令ROM和数据RAM替换为异步

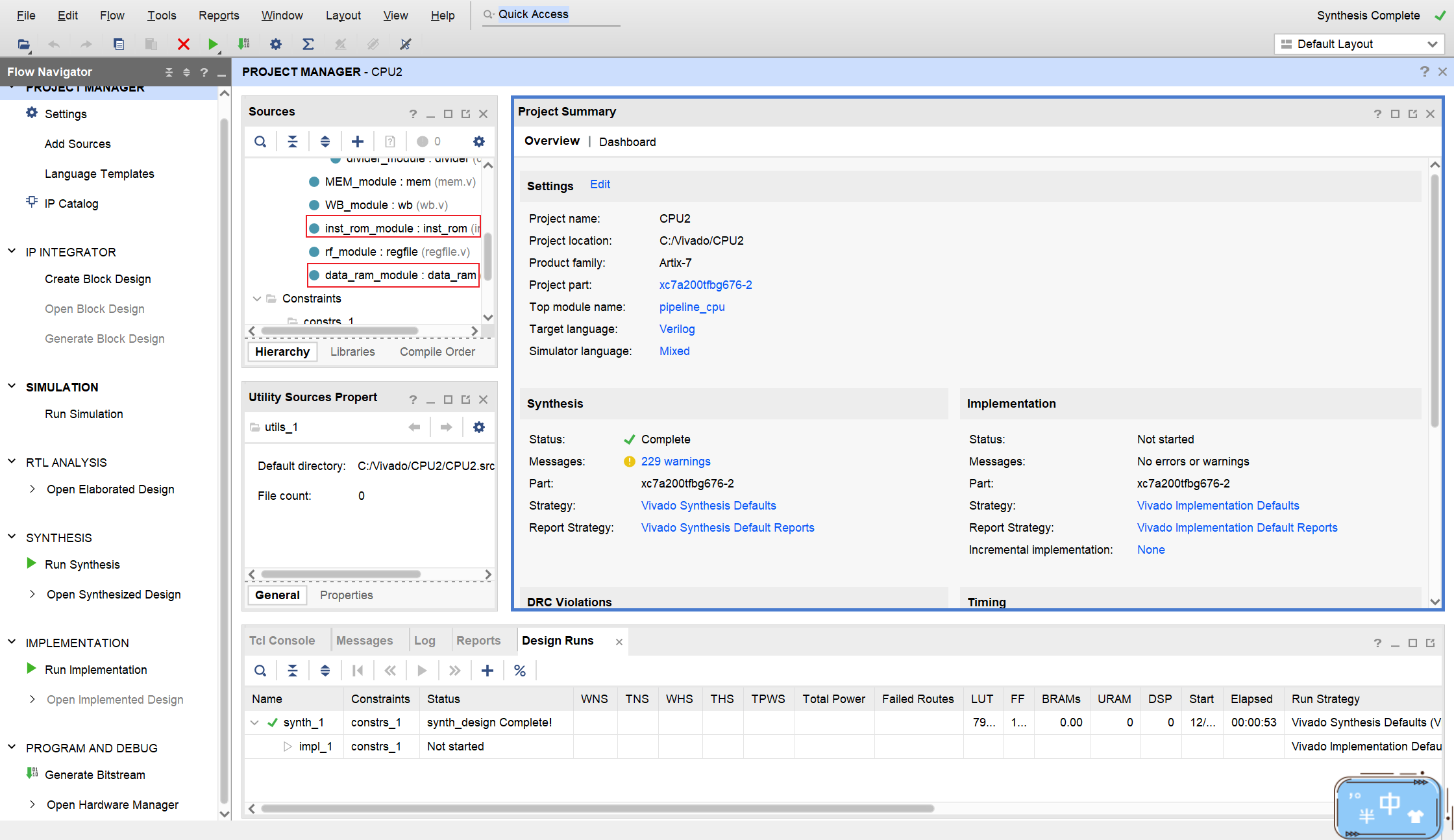
在参考往届学长学姐的报告里，发现当前使⽤的同步指令ROM在取数据时有两拍的延时，即在给出了指令地址的下两拍时钟才能得到对应的指令，但是，当前每⼀个流⽔阶段的执⾏时间是两拍，当同步指令ROM取得数据时，该流⽔阶段正好已经结束了，为了能正确地取得指令，将每⼀个流⽔阶段的执⾏时间延⻓为三拍



为了优化流水线，我们通过将同步指令ROM替换为异步指令ROM来使得指令ROM能在⼀拍内取得数据。（代码指导手册里有，这里不详细讲）

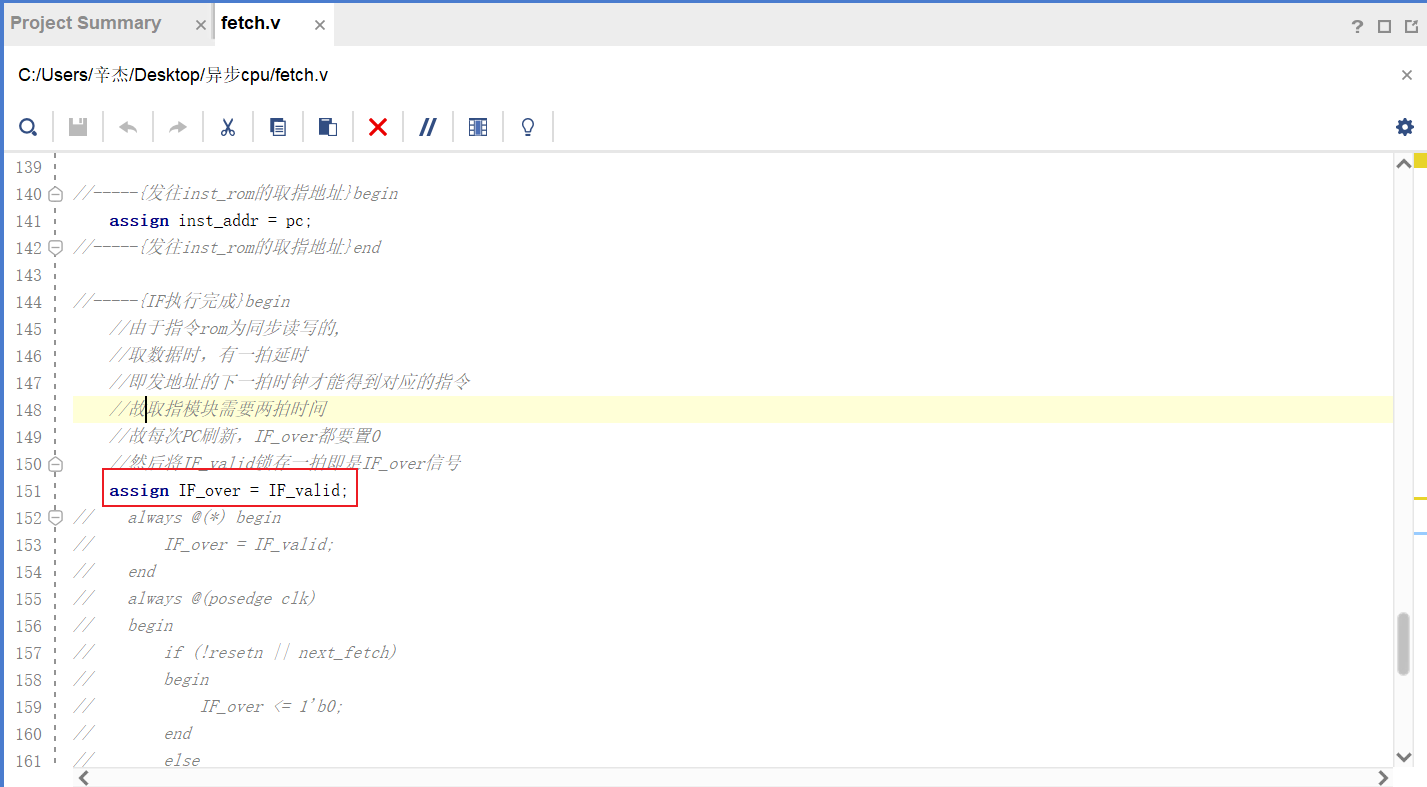


直接加入inst\_rom.v和data\_ram.v文件，不用生成IP核

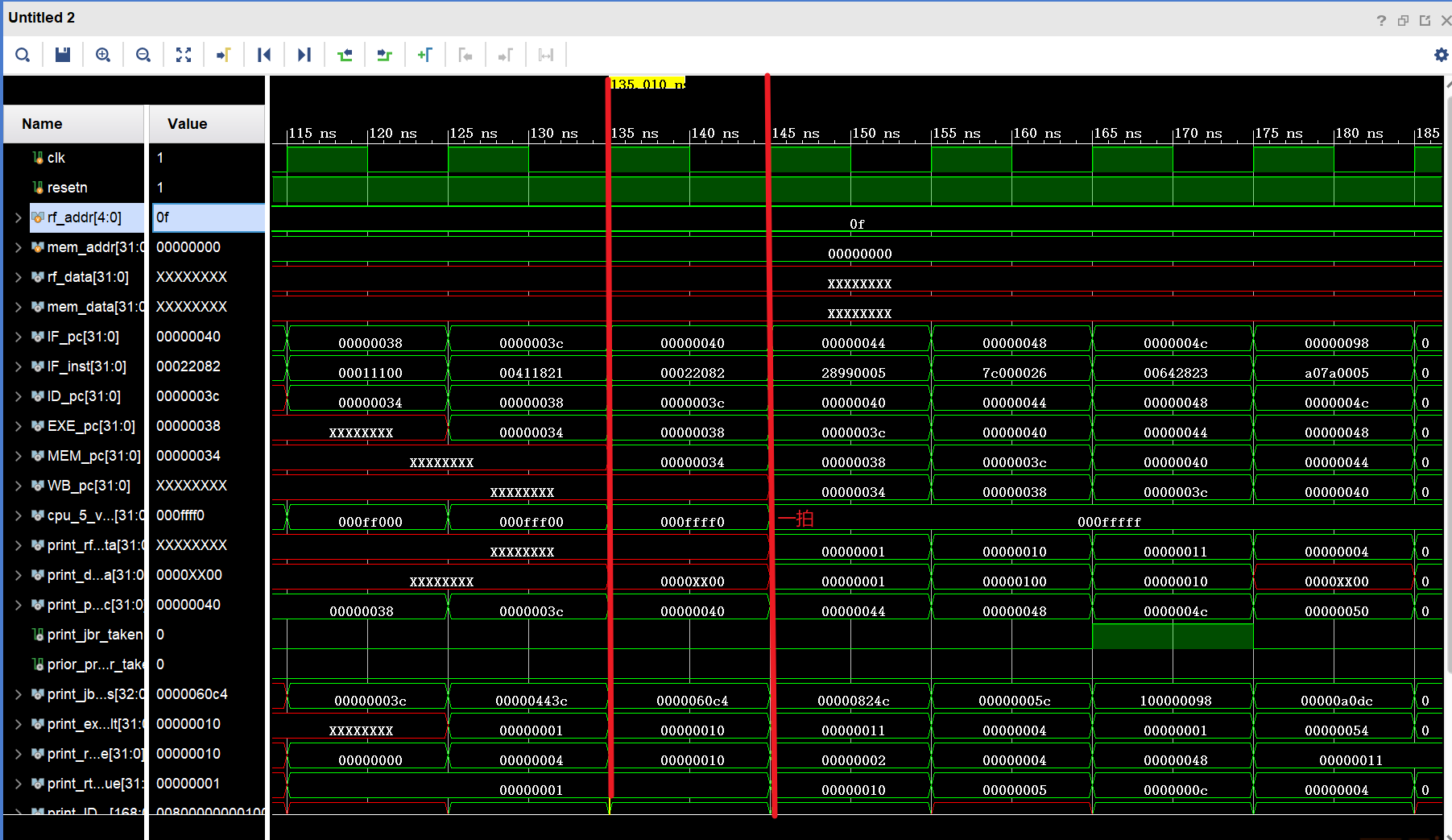


由于此时的指令ROM已被更换为异步读的类型，因此⼀旦进⼊取指阶段便可⽴刻读出指令ROM中的内容，取值阶段实际上在进⼊取值阶段时便已经完成了，⽆需再额外等待⼀拍。

在fetch,v中，将IF\_over改为wire类型。



修改完后即可一拍取得指令，流水线各个阶段也一拍可以解决



## 四、实验内容

### 1、旁路技术

在decode中可以看出，上次实验的五级流水线通过阻塞机制实现解决数据一致性问题。

//-----{ID执行完成}begin  
 //由于是流水的，存在数据相关  
 wire rs\_wait;  
 wire rt\_wait;  
 assign rs\_wait = ~inst\_no\_rs & (rs!=5'd0)  
 & ( (rs==EXE\_wdest) | (rs==MEM\_wdest) | (rs==WB\_wdest) );  
 assign rt\_wait = ~inst\_no\_rt & (rt!=5'd0)  
 & ( (rt==EXE\_wdest) | (rt==MEM\_wdest) | (rt==WB\_wdest) );

准备增加旁路，首先添加输入端口，这些总线数据用于从 **EX/MEM** 和 **MEM/WB** 阶段中提取旁路数据

input [153:0] EXE\_MEM\_bus,  
input [117:0] MEM\_WB\_bus,

增加⽤于参与计算的rs\_value和rt\_value

wire [31:0] correct\_rs\_value;  
wire [31:0] correct\_rt\_value;

增加旁路控制信号

wire [1:0] MX\_bypass; // bit 0 for rs, bit 1 for rt  
wire [1:0] WX\_bypass; // bit 0 for rs, bit 1 for rt

MEM-EX旁路

assign MX\_bypass = {~inst\_no\_rt & (rt!=5'd0) & (rt==EXE\_wdest),  
 ~inst\_no\_rs & (rs!=5'd0) & (rs==EXE\_wdest)};

WB-EXE旁路

assign WX\_bypass = { ~inst\_no\_rt & (rt!=5'd0) & (rt==MEM\_wdest) & ~MX\_bypass[1],  
 ~inst\_no\_rs & (rs!=5'd0) & (rs==MEM\_wdest) & ~MX\_bypass[0]};

最终数据选择

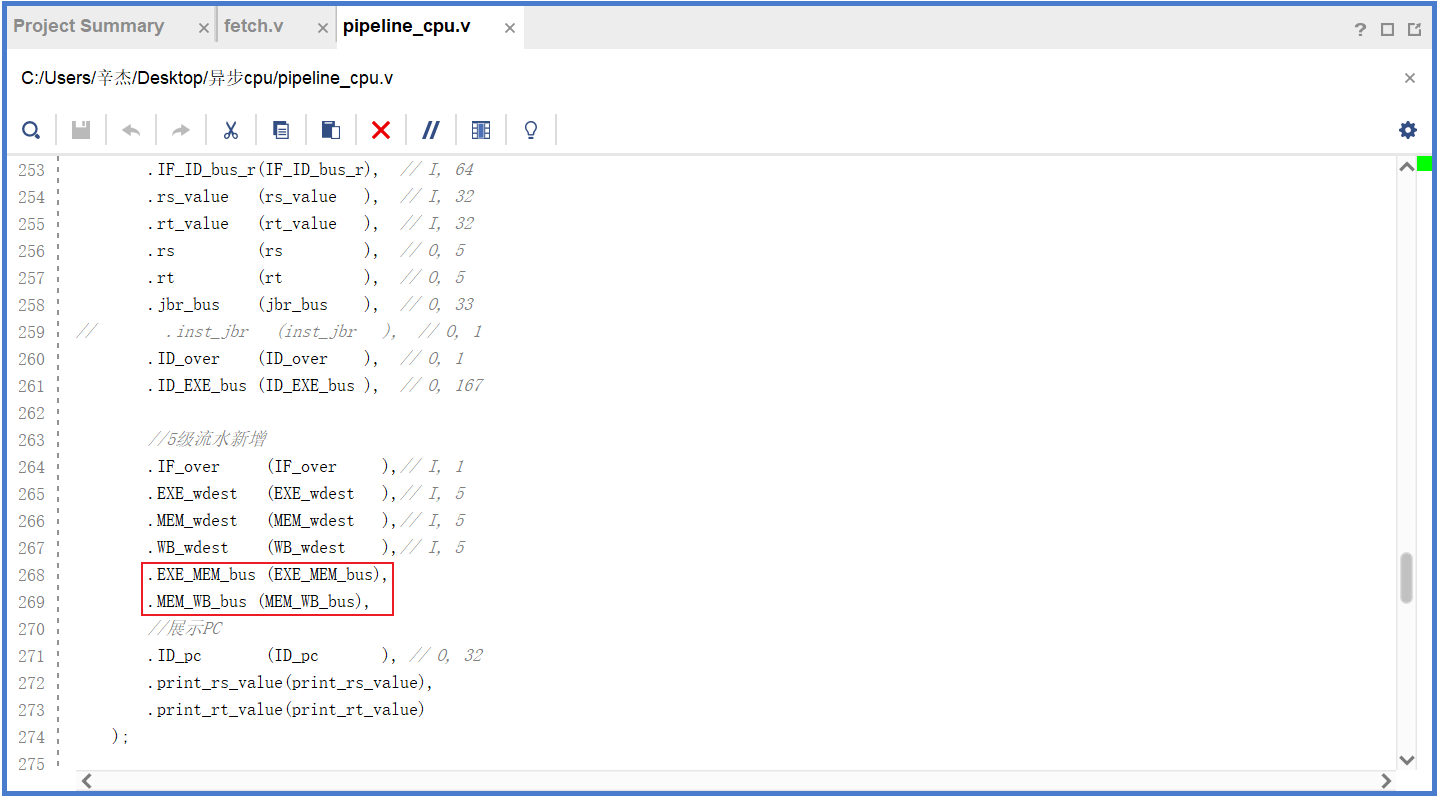
assign correct\_rs\_value = WX\_bypass[0] ? MEM\_WB\_bus[111:80] : (MX\_bypass[0]  
 ? EXE\_MEM\_bus[117:86] : rs\_value);  
assign correct\_rt\_value = WX\_bypass[1] ? MEM\_WB\_bus[111:80] : (MX\_bypass[1]  
 ? EXE\_MEM\_bus[117:86] : rt\_value);

将 rs\_value 和 rt\_value 替换为 correct\_rs\_value 和 correct\_rt\_value

最终，在旁路机制完全覆盖的情况下，不需要等待数据，因此直接将 rs\_wait 和 rt\_wait 赋值为 0，表示流水线无须暂停。

assign rs\_wait = 0;  
assign rt\_wait = 0;

此外在pipeline\_cpu.v向decode模块新增的输⼊端⼝传递数据即可完成旁路



### 2、指令拓展

#### 1）R型指令

为了扩展现有的 R 型指令集，添加了 **无符号除法指令** 和 **无符号取模指令**。由于指令集中没有相关操作，需要设计一个高效的 **无符号除法器**，并对流水线中相关模块进行修改。

##### 无符号除法器设计

无符号除法器实现了逐位除法的功能，包含三个状态：

1. **IDLE**（空闲状态）：等待除法操作的开始信号。
2. **WORK**（运算状态）：通过逐位移位比较的方式执行除法。
3. **DONE**（完成状态）：计算结果输出后返回 IDLE 状态，等待下一次操作。

模块接口定义如下：

module divider (  
 input clk, // 时钟信号  
 input div\_begin, // 除法开始信号  
 input [31:0] div\_op1, // 被除数  
 input [31:0] div\_op2, // 除数  
 output reg [31:0] quotient, // 商  
 output reg [31:0] remainder,// 余数  
 output reg div\_end // 除法结束信号  
);

##### 核心逻辑：逐位除法

在运算状态下，除法器通过以下步骤逐步完成除法：

1. 从被除数中逐位提取最高位加入余数寄存器。
2. 比较当前余数和除数：
   * 若余数大于等于除数，更新余数为当前余数减去除数，并将当前位商设置为 1。
   * 否则，将当前位商设置为 0。
3. 商寄存器左移一位，将当前位商拼接到商寄存器。
4. 更新被除数，循环处理直至完成所有位的计算。

代码实现如下：

WORK: begin  
 if (bit\_index >= 0) begin  
 // 逐位操作：更新余数和商  
 temp\_remainder = {temp\_remainder[30:0], dividend[31]};  
 dividend = dividend << 1;  
  
 if (temp\_remainder >= divisor) begin  
 temp\_remainder = temp\_remainder - divisor;  
 temp\_quotient = (temp\_quotient << 1) | 1;  
 end else begin  
 temp\_quotient = temp\_quotient << 1;  
 end  
  
 // 检查是否处理完成  
 if (bit\_index == 0) begin  
 state <= DONE;  
 end else begin  
 bit\_index <= bit\_index - 1;  
 end  
 end  
end

##### 无符号除法指令实现

###### 无符号除法指令格式

无符号除法指令的 funct 字段设置为 6'b011001，表示该指令的功能为无符号除法。

31 26 25 21 20 16 15 11 10 6 5 0  
+---------+--------+--------+--------+--------+--------+  
| 000000 | rs | rt | rd | 00000 | 011001 |  
+---------+--------+--------+--------+--------+--------+  
 6 5 5 5 5 6

以下是实现流程：

###### 1. 扩展 ID 到 EXE 的总线

为传递控制信号，扩展了 ID 阶段到 EXE 阶段的总线：

wire [167:0] ID\_EXE\_bus; // 原为 [166:0]  
reg [167:0] ID\_EXE\_bus\_r;

###### 2. 定义控制信号

在 decode.v 文件中，添加无符号除法指令的识别逻辑，并将控制信号通过总线传递：

wire inst\_DIV; // 无符号除法指令标识  
assign inst\_DIV = op\_zero & sa\_zero & (funct == 6'b011001);  
  
wire div\_enable; // 除法控制信号  
assign div\_enable = inst\_DIV;

###### 3. EXE 阶段处理逻辑

在 EXE 阶段，根据控制信号启动除法器并处理计算结果：

assign div\_begin = div\_enable & EXE\_valid;  
assign EXE\_over = EXE\_valid & (~div\_enable | div\_end);  
  
assign exe\_result = div\_enable ? quotient : alu\_result;

###### 4. 指令格式与执行

设置指令格式为：0000\_0000\_0100\_0100\_1101\_0000\_0001\_1001，对应操作：

DIV $26, $2, $4 # $2 / $4 的结果存储到 $26

例如，寄存器 $2 的值为 16，寄存器 $4 的值为 4，最终 $26 的结果为 4。

##### 无符号取模指令实现

###### 无符号取模指令格式

在无符号除法的基础上，进一步实现无符号取模指令，其 funct 字段为 6'b011010。

31 26 25 21 20 16 15 11 10 6 5 0  
+---------+--------+--------+--------+--------+--------+  
| 000000 | rs | rt | rd | 00000 | 011010 |  
+---------+--------+--------+--------+--------+--------+  
 6 5 5 5 5 6

主要修改如下：

###### 1. 扩展总线

再次扩展 ID 阶段到 EXE 阶段的总线：

wire [168:0] ID\_EXE\_bus; // 原为 [167:0]  
reg [168:0] ID\_EXE\_bus\_r;

###### 2. 定义取模控制信号

在 decode.v 文件中，添加取模指令识别逻辑：

wire inst\_MOD; // 无符号取模指令标识  
assign inst\_MOD = op\_zero & sa\_zero & (funct == 6'b011010);  
  
wire mod\_enable; // 取模控制信号  
assign mod\_enable = inst\_MOD;

###### 3. EXE 阶段处理逻辑

在 EXE 阶段，根据控制信号选择输出结果：

assign div\_begin = (div\_enable | mod\_enable) & EXE\_valid;  
assign EXE\_over = EXE\_valid & (~div\_enable & ~mod\_enable | div\_end);  
  
assign exe\_result = mod\_enable ? remainder : (div\_enable ? quotient : alu\_result);

###### 4. 指令格式与执行

设置指令格式为：0000\_0000\_0110\_0010\_1101\_1000\_0001\_1010，对应操作：

MOD $27, $3, $2 # $3 % $2 的结果存储到 $27

例如，寄存器 $3 的值为 17，寄存器 $2 的值为 16，最终 $27 的结果为 1。

通过添加无符号除法和取模指令，并实现对应的硬件逻辑，扩展了 R 型指令的功能。整个实现包括除法器设计、流水线控制信号处理、EXE 阶段结果选择等模块化设计，确保了系统的扩展性和功能完整性。

#### 2）J型指令

由于实验中已经实现了 **直接跳转指令 (J target)** 和 **跳转并链接指令 (JAL target)**，为进一步完善功能，这里设计了一个新的指令 **JALX**，用于实现跳转链接到 PC+12 的位置。JALX 指令的 op 字段被设置为 011111，其格式如图所示：

##### JALX 指令格式

31 26 25 0  
+---------+-----------------------------+  
| 011111 | target |  
+---------+-----------------------------+

* **op**: 6 位操作码，固定为 011111
* **target**: 26 位目标地址

###### 实现步骤

###### 修改 decode.v 文件

在 decode.v 中添加对 JALX 指令的解析和控制信号定义：

// 定义 JALX 指令  
wire inst\_JALX;  
assign inst\_JALX = (op == 6'b011111);  
  
// 更新分支与跳转相关控制信号  
assign inst\_jbr = inst\_J | inst\_JAL | inst\_jr |   
 inst\_BEQ | inst\_BNE | inst\_BGEZ |   
 inst\_BGTZ | inst\_BLEZ | inst\_BLTZ | inst\_JALX;  
  
// 更新写目标寄存器信号  
assign inst\_wdest\_31 = inst\_JAL | inst\_JALX;  
  
// 更新不需要 rt 数据的指令信号  
assign inst\_no\_rt = inst\_ADDIU | inst\_SLTI | inst\_SLTIU |  
 inst\_BGEZ | inst\_load | inst\_imm\_zero |  
 inst\_J | inst\_JAL | inst\_MFC0 |  
 inst\_SYSCALL | inst\_DIVI | inst\_MODI | inst\_JALX;

###### 操作数设置

在 decode.v 文件中更新 ALU 操作数选择逻辑，支持 JALX 指令：

// 设置第一个操作数  
assign alu\_operand1 = inst\_j\_link ? pc :  
 inst\_JALX ? pc :  
 inst\_shf\_sa ? {27'd0, sa} : correct\_rs\_value;  
  
// 设置第二个操作数  
assign alu\_operand2 = inst\_j\_link ? 32'd12 :  
 inst\_JALX ? 32'd4 :  
 inst\_imm\_zero ? {16'd0, imm} :  
 inst\_imm\_sign ? {{16{imm[15]}}, imm} : correct\_rt\_value;

###### 更新指令内存

在指令存储器中添加 JALX 指令，将地址 48H 处设置为：

assign inst\_rom[18] = 32'h7C000026; // 0111\_1100\_0000\_0000\_0000\_0000\_0010\_0110

* **指令含义**: jalx $31, target，表示跳转链接到目标地址并将 PC+12 的值存入 $31。
* **测试设置**: 将跳转目标地址映射到 98H 的位置，并在寄存器 $31 中存储跳转链接值 54H。

#### 3）I型指令

在实现了 R 型的无符号除法和取模指令之后，进一步扩展指令集，添加 **无符号立即数除法** 和 **无符号立即数取模指令**。这两类指令的设计无需对流水线的总线结构进行拓展，仅需调整指令解码逻辑即可。

##### 无符号立即数除法指令格式

31 26 25 21 20 16 15 0  
+---------+--------+--------+-----+-----+  
| 011001 | rs | rt | imm |   
+---------+--------+--------+-----+-----+  
 6 5 5 16

##### 无符号立即数取模指令格式

31 26 25 21 20 16 15 0  
+---------+--------+--------+-----+-----+  
| 011010 | rs | rt | imm |   
+---------+--------+--------+-----+-----+  
 6 5 5 16

##### 解码逻辑修改

为了支持无符号立即数除法和取模指令，在 decode.v 文件中添加对应的控制信号，定义 inst\_DIVI 和 inst\_MODI 用于判断是否为相应的指令。

###### 1. 定义控制信号

根据指令的操作码 op 字段设置判断逻辑：

wire inst\_DIVI, inst\_MODI;  
  
// 判断是否为无符号立即数除法或取模指令  
assign inst\_DIVI = (op == 6'b011001);  
assign inst\_MODI = (op == 6'b011010);

###### 2. 更新立即数扩展逻辑

对于立即数操作，需要进行零扩展。定义 inst\_imm\_zero 用于标识需要零扩展的指令：

assign inst\_imm\_zero = inst\_ANDI | inst\_LUI | inst\_ORI | inst\_XORI | inst\_DIVI | inst\_MODI;

###### 3. 更新目标寄存器选择逻辑

无符号立即数除法和取模指令的结果存储在目标寄存器 rt 中，需要更新控制信号 inst\_wdest\_rt：

assign inst\_wdest\_rt = inst\_imm\_zero | inst\_ADDIU | inst\_SLTI | inst\_SLTIU | inst\_load | inst\_MFC0 | inst\_DIVI | inst\_MODI;

###### 4. 确认指令是否需要 rt

更新 inst\_no\_rt 信号，用于识别是否需要从寄存器堆读取 rt：

assign inst\_no\_rt = inst\_ADDIU | inst\_SLTI | inst\_SLTIU | inst\_BGEZ | inst\_load | inst\_imm\_zero  
 | inst\_J | inst\_JAL | inst\_MFC0 | inst\_SYSCALL | inst\_DIVI | inst\_MODI;

###### 5. 合并除法和取模控制信号

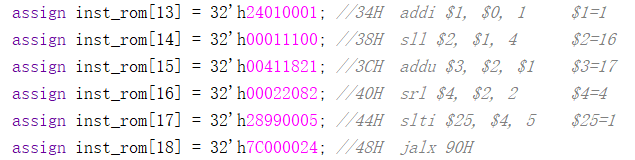
为了简化后续处理，将除法和取模指令的控制信号合并：

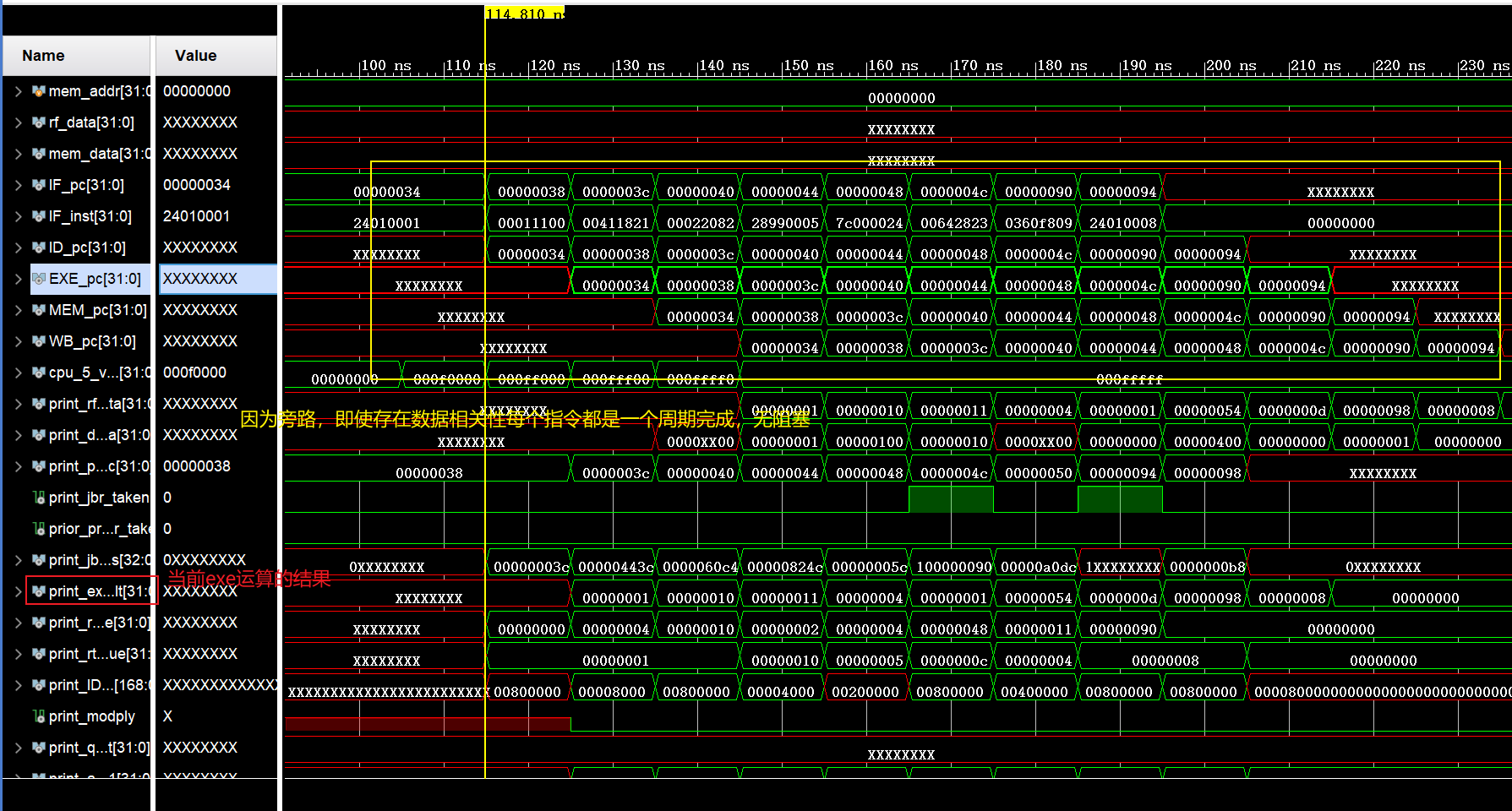
assign divply = inst\_DIV | inst\_DIVI; // 包含 R 型和 I 型除法指令  
assign modply = inst\_MOD | inst\_MODI; // 包含 R 型和 I 型取模指令

通过以上修改，成功扩展了指令集，支持无符号立即数除法和取模指令，同时保持了流水线的结构简洁性和扩展性。

## 五、程序展示

### 1、旁路技术

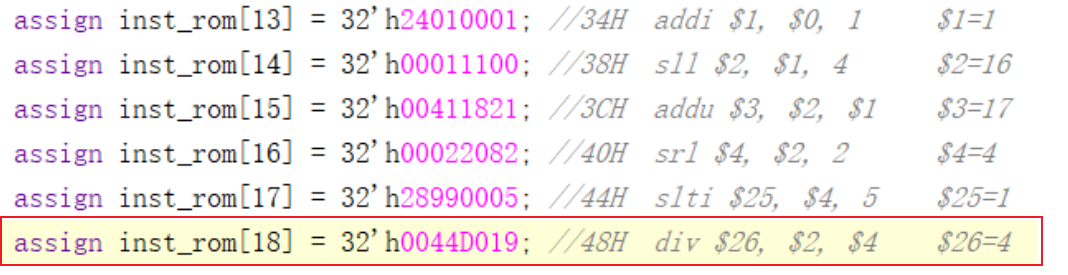




### 2、指令拓展

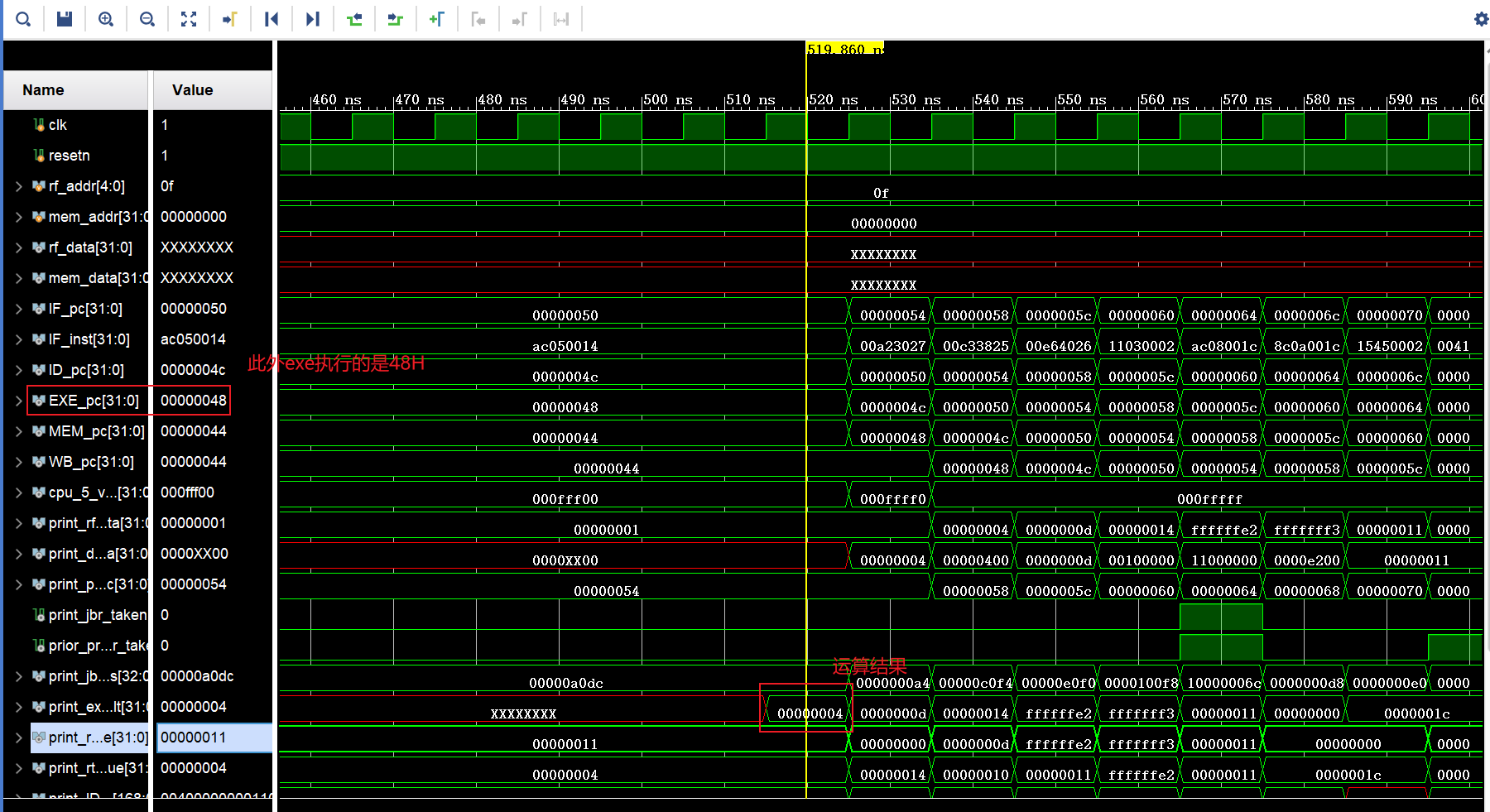
#### 1）R型指令

##### 无符号除法

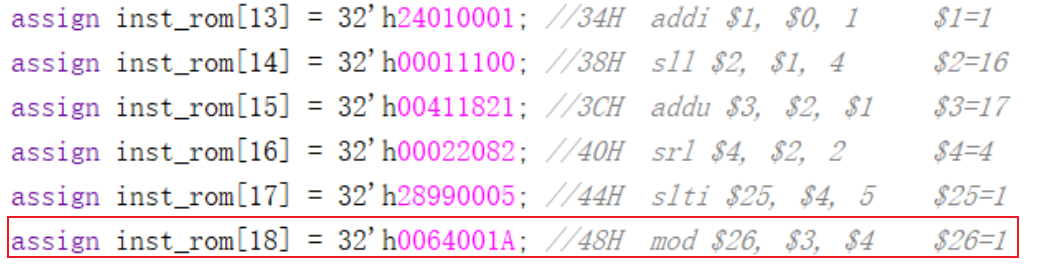


32'h0044D019展开为：

op (6) rs (5) rt (5) rd (5) shamt (5) funct (6)  
000000 00010 00100 11010 00000 011001

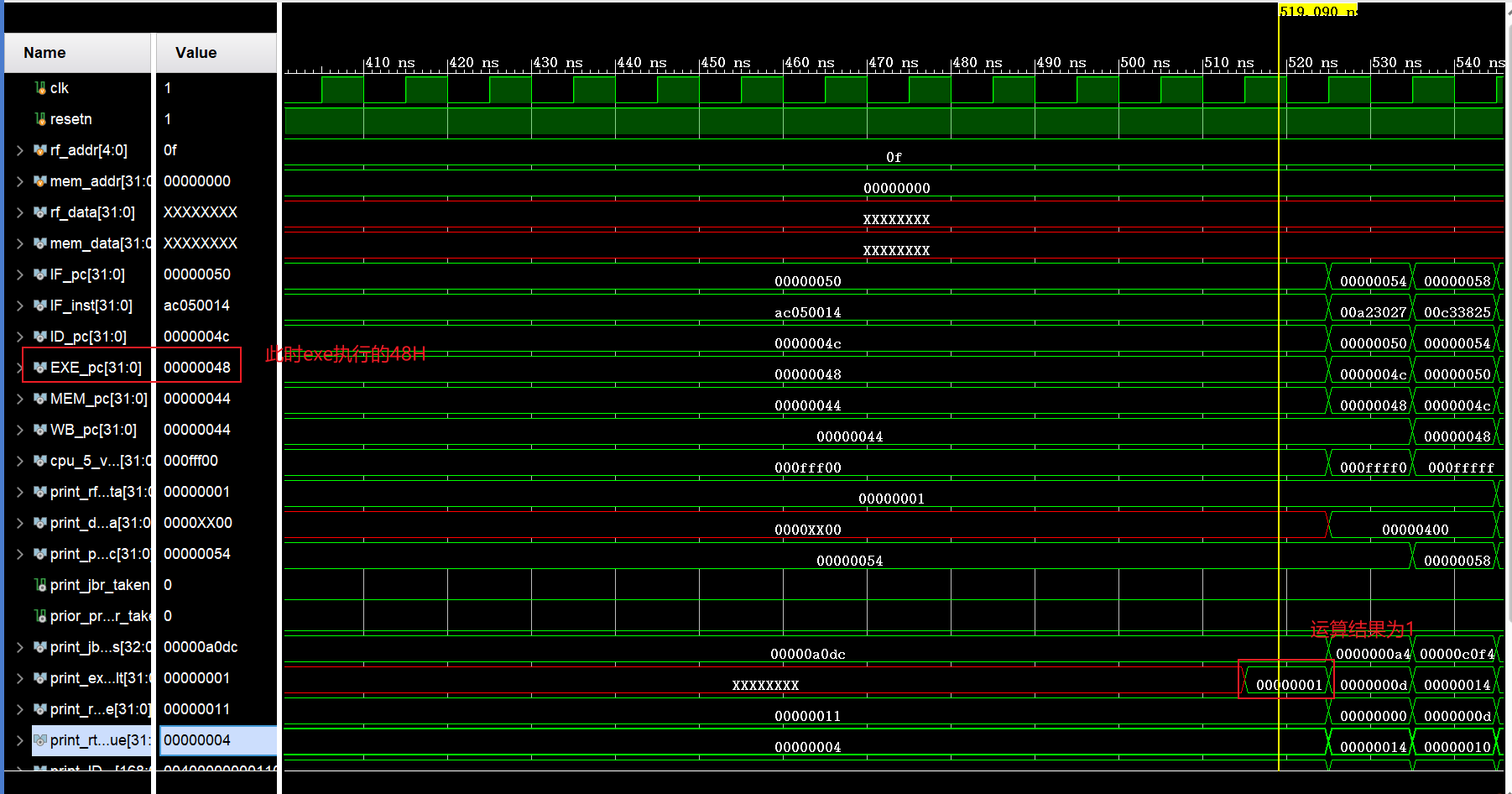


##### 无符号取模

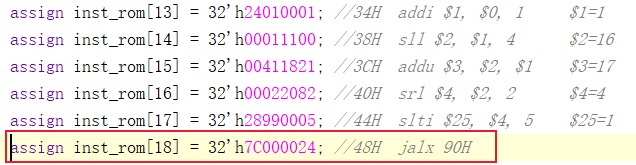


32’h0064001A展开如下

op (6) rs (5) rt (5) rd (5) shamt (5) funct (6)  
000000 00011 00100 11010 00000 011010

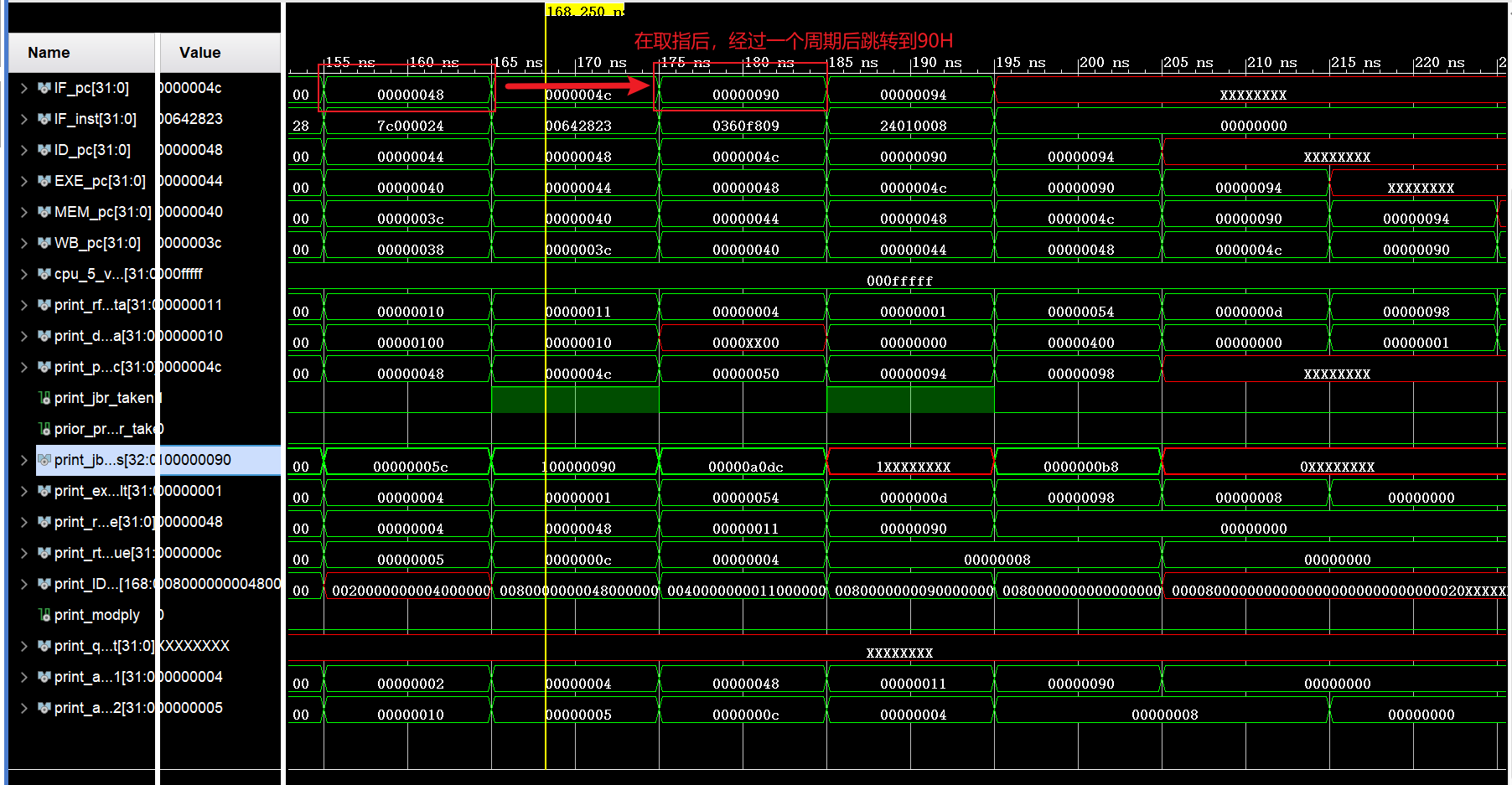


#### 2）J型指令



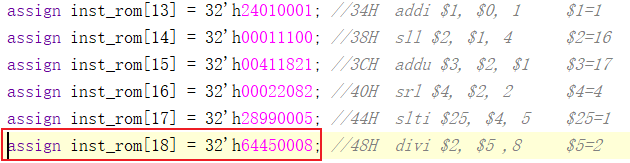
32’h7C000024展开如下

op (6) target(25)  
011111 0 0000 0000 0000 0000 0010 0100



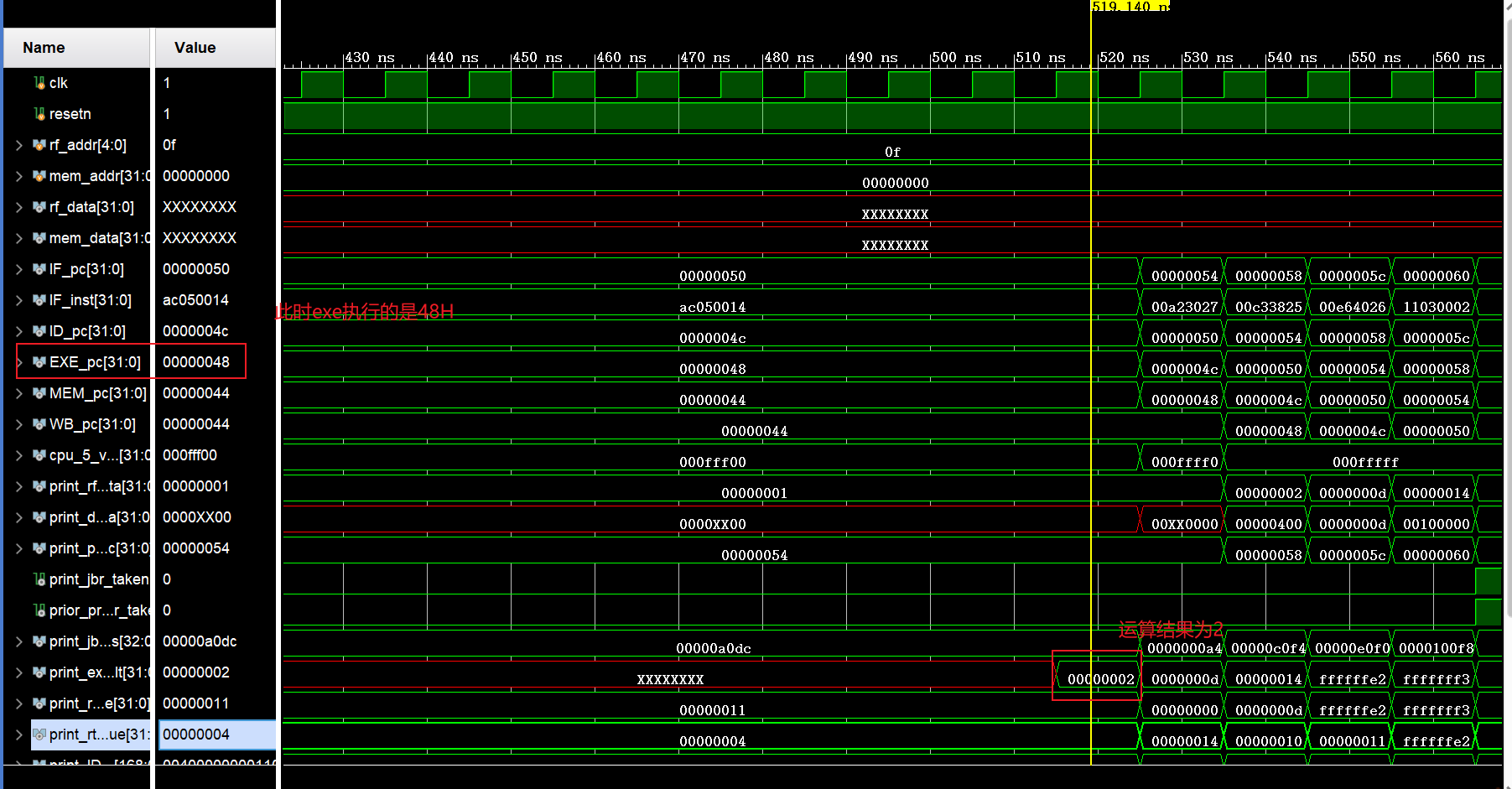
#### 3）I型指令

##### 无符号立即数除法

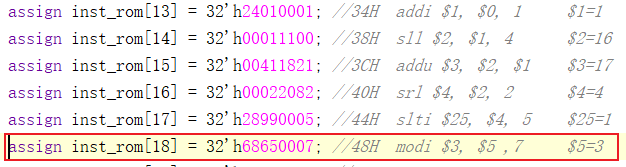


32h’64450008展开为

op (6) rs (5) rt (5) imm(16)  
011001 00010 00101 0000 0000 0000 1000

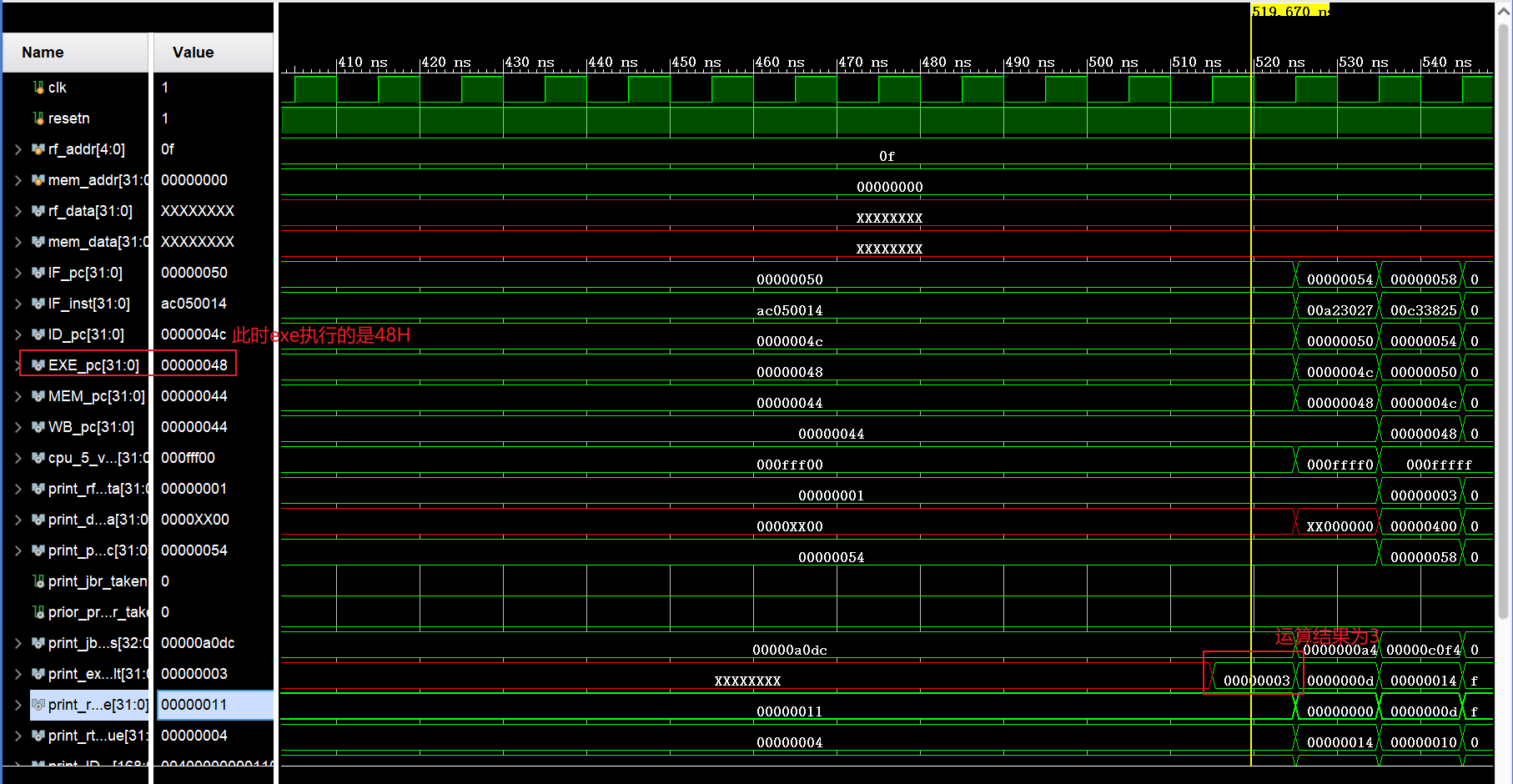


##### 无符号立即数取模



32’h68650007展开如下

op (6) rs (5) rt (5) imm(16)  
011010 00011 00101 0000 0000 0000 0111



## 六、思想感悟

这次实验任务量较为繁重，即使前期时间充足但也无从下手，最后还是只能完成相对简单的1和2，主要完成的是将同步改为异步，这样就可以每个阶段正常情况都是一步完成，异步也能完成旁路，当是多拍在现实流水线中代价略大。指令拓展也有Verilog基础相对容易一点，主要是增加了一个除法器。

通过实验，深刻地理解了计算机组成原理和体系结构的理论知识。这种从书本到实践的过程，将抽象的概念具体化，加深了对CPU设计的理解。